1970

Fault detection of microcircuits with graphical evaluation and review technique

Lucius J. Riccio
Lehigh University

Fault Detection of Microcircuits with Graphical Evaluation and Review Technique (GERT)

by
Lucius J. Riccio

A Thesis

Presented to the Graduate Faculty

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Industrial Engineering

Lehigh University

1970

CERTIFICATE OF APPROVAL

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

May 18, 1970
Date

Gary E Whitehouse
Professor in Charge

Arthur F. Gower
Head of the Department
of Industrial Engineering

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to the members of the Industrial Engineering Department who have made my graduate study both possible and pleasurable.

In particular, I would like to thank Dr. Gary E. Whitehouse whose generous advice and firm guidance has been distinctively magnanimous.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The purpose of this thesis is to demonstrate that electronic microcircuits can be modeled using GERT, Graphical Evaluation and Review Technique. It is shown that microcircuits can be modeled and simulated quickly and easily with the aid of NASA's GERTS II Program. The value of this skill is clearly visable to fault analysis. The GERTS II Simulator is a Fortran-Based Program and can be easily adapted and/or modified. Thus, it would be a simple task to develop a universal computer program that would accept data concerning any logic circuit and analyze its fault characteristics. The input formats would be simple since the GERT's program could provide the mechanism to bridge the interface between electronic network and GERT network. The output would be a matter of data processing.

Thus, the application of GERT to the area of fault analysis significantly extends the value of many detection procedures. The development of this GERT-Based Program would not only be important to present fault algorithms but also would provide a tool to aid in the continuing research for new algorithms.

INTRODUCTION

Fault detection of electronic microcircuits is a relatively new area of research. The essential objective of this research is to develop algorithms to detect and distinguish between failures in logic circuits. Logic electronic circuits are conceptually characterized by logical gates and connectives which operate on electronic impulses in a prescribed Boolean function. These circuits are often quite complex yet at the same time quite small, thereby creating a massive detection problem. However, the value of fault information to such fields as quality control and reliability engineering easily warrants the investigation of new methods of tackling this problem.

Recent research has yielded several algorithms to investigate the operation of logic circuits which have been internally befuddled by hardware failures. The nature of these algorithms, in conjunction with the size of the circuits, restrict their application solely to computer manipulation. Thus, the ability to model circuits is a worthy ambition within the purview of one clearly salient pursuit. The realization of the true effacacy of many well-founded fault

detection techniques is to a great extent dependent on the state of the art of programming such algorithms. Therefore, accompanying the research to find new and better methods of analysis, there must be parallel research to develop new and better computer techniques to implement fault detection procedures.

Due to the nature of microcircuits, line values may be emperically imposed and recorded only on the input and output terminals of a circuit. Therefore, any algorithms to diagnose faults must yield a solution that will "invisibly" investigate the internal peculiarities of the circuit by generating information concerning input-output combinations. The advantage of graphical representation of the circuit for analysis is obvious. With the aid of a network diagram the internal operations can be altered. The consequence of the alteration of proper operation can then be traced to and registered at the output terminals. This approach will yield the input-output information needed to diagnose faults within a circuit.

The graphical representation of electronic networks by GERT will provide a foundation for implementing the approach discussed in the preceding paragraph. The GERT network will

yield precisely the information required to analyze a circuit. The intent is to develop a GERT technique that will quickly and easily model any logic circuit and perform simulated fault tests leading to effective check-out procedures.

Electronic Logic Circuits

Microcircuits consist of logical gates and line connectives. The gates are comprised of various electronic elements such as resistors, diodes, and transistors and when several gates are connected and placed together on a "chip", the system is called a microcircuit. However, for our purposes a logic gate can be considered to be a "black box" apparatus and can be symbolically represented as such.

An electronic gate functionally alters electronic impulses in accordance with the Boolean logic indigenous to that gate. The logic functions have been well defined and suitably titled. For example, the logic of an "AND" gate, in truth table form, is listed in Figure I. That is, the "AND" gate will yield an output of "1" only when all inputs are at the level "1", and it will yield a "0" at all other times. Other gate types are "OR", "NAND" (not and), "NOT" (invertor), and "EXCLUSIVE-OR". The

"0"s and "1"s in the table represent two levels of operation of the circuit with respect to an electronic parameter. For example, they may represent two levels of voltage.

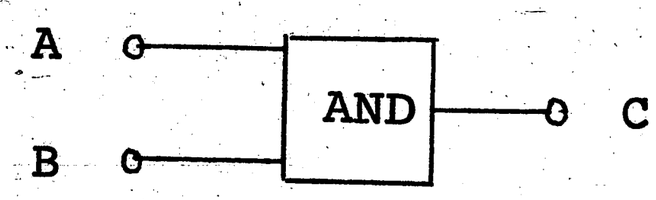
When many gates are joined together, they comprise a microcircuit. A simple microcircuit is shown in Figure II. This diagram is the symbolic representation of a half-adder. The circuit performs the Boolean operations on the inputs A and B and yields the desired corresponding outputs C and D.

Types of Logic Circuits

There are two major categories of logic circuits: (1) combinational logic, and (2) sequential logic circuits. A combinational logic circuit is defined as a circuit whose output is solely dependent on the present value of the inputs. The outputs of a sequential circuit are dependent not only on the present value of the inputs but also on the past history of the system. That is, there is some form of "memory" device within the circuit to store past information. Sequential circuits are also divided into two major classes: (1) synchronous, and (2) asynchronous. There is a degree of

The "AND" Gate

Figure I.

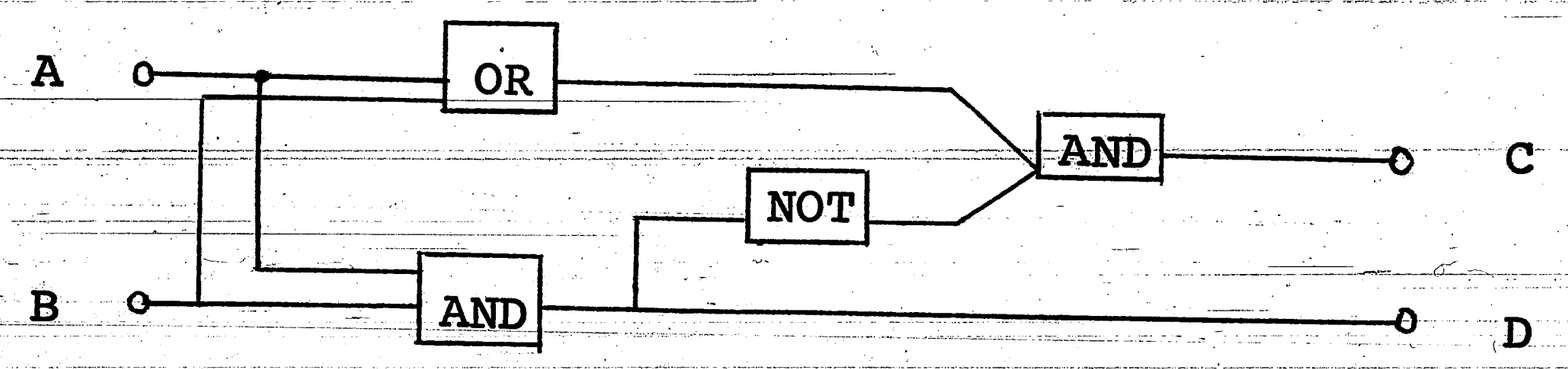


A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Combinational Logic Circuit

Half-Adder

Figure II.



ambiquity associated with these terms due to the lack of an exacting definition. However, this thesis shall use them according to the following interpretations. Synchronous circuits are "clocked" (an impulse is imposed on various gates at periodic intervals) defining precisely when the inputs are to be analyzed, while an asynchronous circuit is either "unclocked" or the clocking action is such that more than one transition of state can occur between clock pulses.

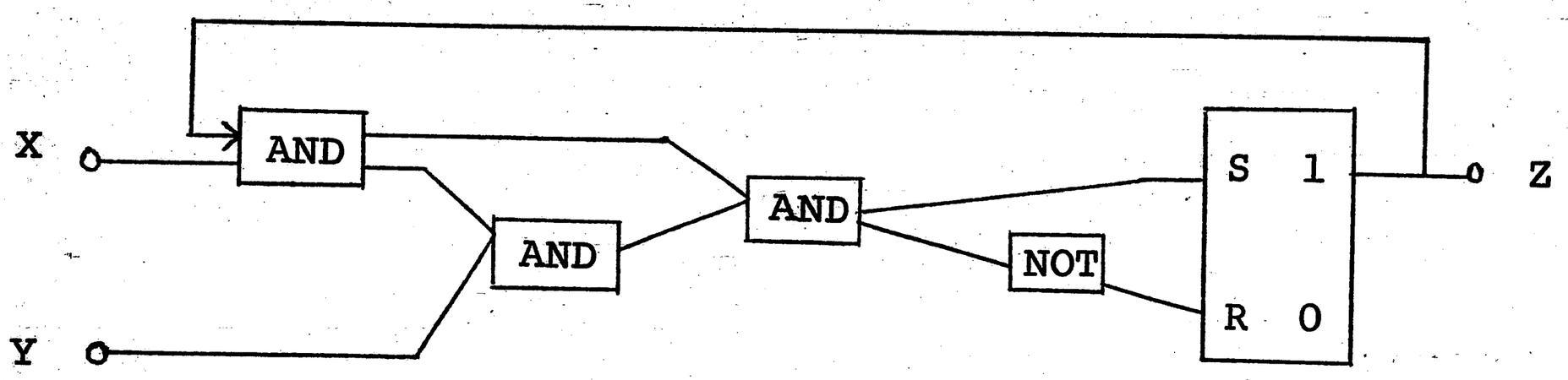
Figure II is an example of a combinational logic circuit. Figure III is an example of an asynchronous switching circuit. The "memory" element is the "set-reset flip-flop".

Figure IV is an illustration of a synchronous sequential circuit. The diagram is a representation of a sequential parity checker. The clock provides an impulse according to a designed periodic pattern.

It is apparent that there is an element of "timing" involved in sequential circuits that is not present in combinational circuits. The difficulty of modeling sequential circuits will certainly be a function of the ability to incorporate this timing facet within the simulation prototype.

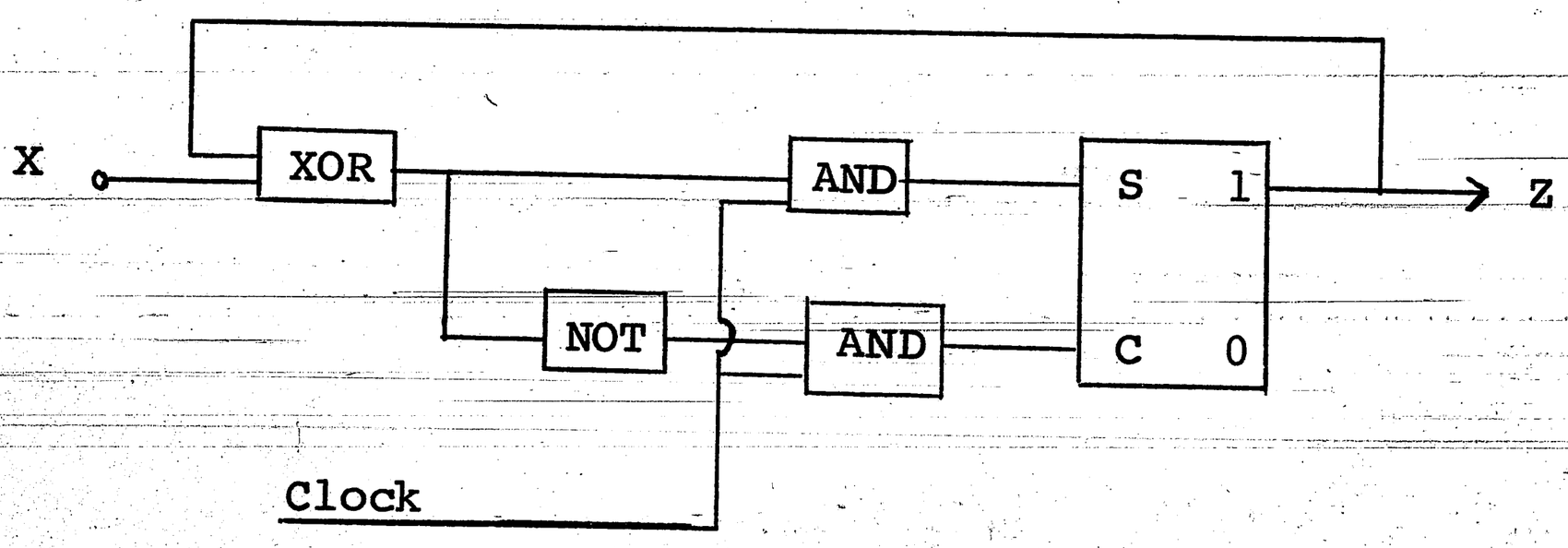
Asynchronous Sequential Logic Circuit

Figure III.



Synchronous Sequential Logic Circuit

Figure IV.



The intention of the preceding discussion was to familiarize the reader with the field of electronic microcircuits. It certainly leaves many questions unanswered. However, as this thesis becomes more specific in its attempt to fulfill its charge, the scope and depth of the problem will become sharper and more discernible.

GERT: Graphical Evaluation and Review Technique

A discussion of GERT is appropriate at this time before the actual modeling of circuits is attempted.

Although GERT was originally developed to aid in the analysis of stochastic networks, it has proven to be a useful research tool when applied to many types of scientific and engineering problems. It can be used to model and solve complex systems of a wide variety. Thus, it is not surprising that this graphical technique could be applied to the analysis of electronic circuits.

GERT is, as the name applies, a graphical or network analyzer. The elements of a GERT network are directed branches and logical nodes. A directed branch has associated with it

one node from which it emanates and one node at which it terminates. The nodes embody logical functions which are dependent upon the application.

There are several versions of GERT, each with an accompanying computer program. In this application the GERTS II Simulator, developed for NASA, will be adapted and implemented.

The GERTS II Simulation Program

The GERTS II Simulation Program is written in Fortran using concepts employed in GASP II A(14). In GERTS II a node is described in terms of two parameters:

1. The number of releases, i.e., the number of activities leading into the node that must be realized before the node can be realized; and

2. A node type which specifies:

   a. Whether the output side of the node is deterministic (all activities emanating from the node are taken) or probabilistic (random sampling is to be performed to

determine which one of the activities emanating from the node is taken); and

b. The number of releases required to have the node realized again.

In this application of GERT all nodes will be deterministic. The number of releases associated with a node specifies the number of times activities incident to the node must be realized before the node can be realized. When the number of releases is 1, the input side of the node can be thought of as an OR operation. If the number of releases equals the number of activities incident to the node, the node can be thought of as an AND operation. However, it is permissible to specify the number of releases to be less than or greater than the number of activities incident to the node.

A GERTS II node is graphically represented in Figure V(a). The node in Figure V(b), node 20 must have two realizations to be released the first time and an infinite number of realizations to be released a second time, which in effect means the node can be released only once.

GERT Node

Figure V.

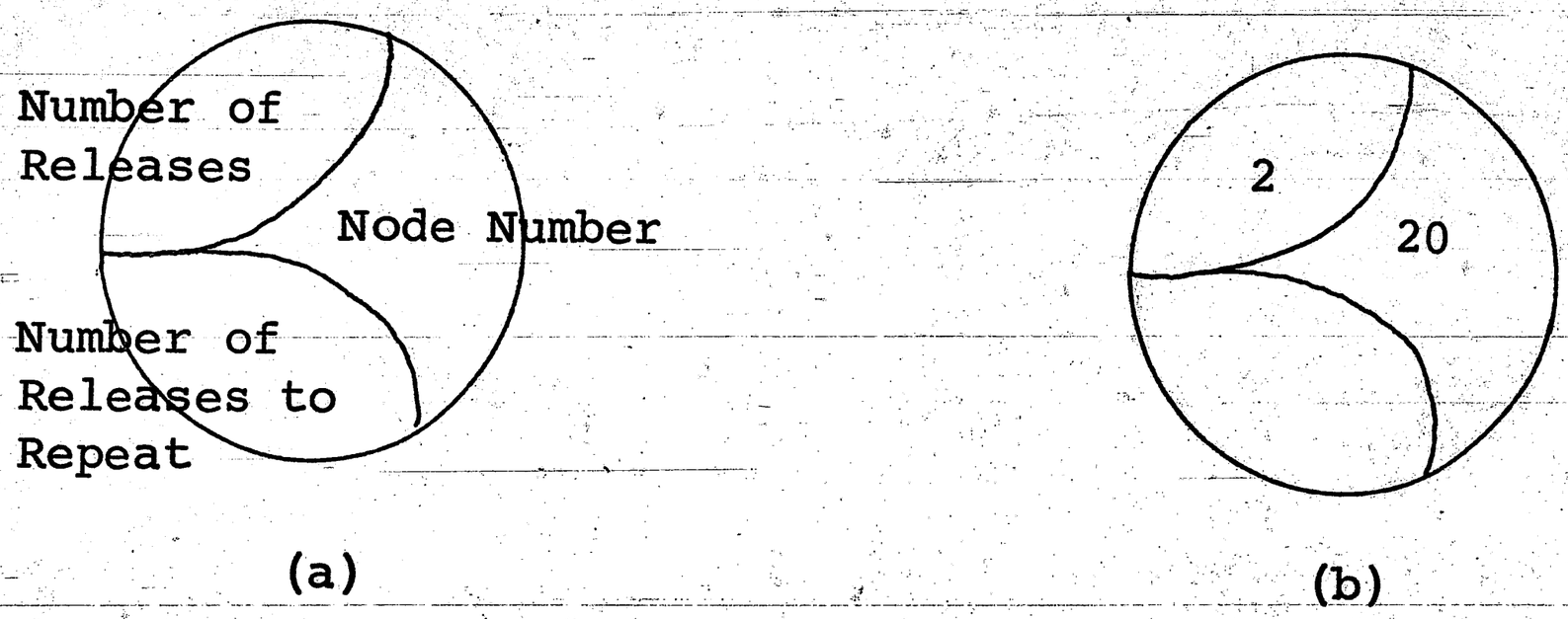
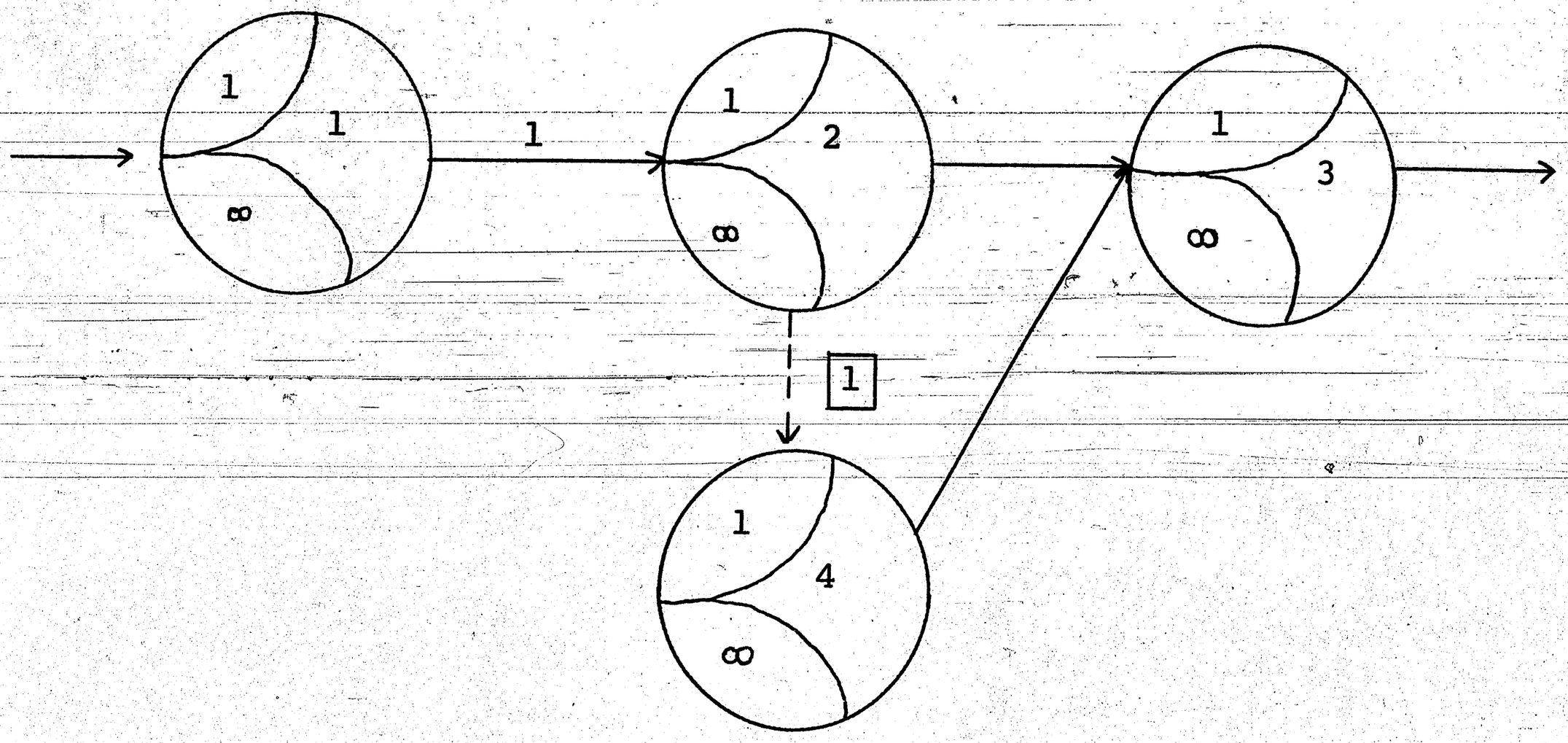


Illustration of Network Modification

Figure VI.



Activity numbers are given to activities (the directed branches of a GERT network) to permit network modifications based on the realization of the activity. Specification of an activity number does not automatically indicate that the network will be modified. However, only activities with activity numbers can cause the network to be modified. Network modification involves the replacing of a node by another node. The node to be replaced is deleted from the network when and if it is realized. The activities then caused to occur are from the node that is inserted.

Figure VI is an example of a network incorporating the network modification option. Associated with the path from node 1 to node 2 is activity number 1 and associated with that activity number is branch [1], represented by a dotted line. The modification is read: If the activity from node 1 to node 2 is realized, node 4 replaces node 2.

With these added features of GERTS II, it is now possible to model electronic circuits.

MODELING LOGIC CIRCUITS WITH GERT

General Concepts

As was shown earlier an electrical node or gate operates upon electronic parameters of a circuit in a Boolean function. This operation can be represented by a truth table notation. For example, the Boolean function of an "AND" gate can be put in truth table form as shown in Figure I. The "zeros" and "ones" in the table represent two levels of operation of the circuit with regard to an electrical parameter. For example, they may represent two levels of voltage. Then the "AND" gate represents a logical operation that yields an output voltage of "1" only when all input voltages are at that same value. The output of the gate will be constant and will remain the same value until the input characteristics have been changed.

As shown earlier, a GERT node operates in a similar manner. If the input values sufficiently satisfy the GERT specifications of the node, the GERT node will activate an impulse on the output branches. If the specifications are not met the output branches will remain dormant. With this understanding of the operations of electrical gates and GERT nodes, the following

analogy can be made. A GERT node can be used to represent an electronic logic gate by allowing the realization of an activity (directed branch) of a GERT network to be equivalent to the value "1" assigned to a gate connector (line) of an electronic circuit. That is if an activity of a GERT network is realized, then the corresponding line in the electronic circuit that has been modeled in GERT has a truth value function of "1". Likewise, if the GERT activity is not realized the corresponding line in the electronic circuit has a value of "0". With this definition, circuits can be modeled and their internal operations inspected by GERT.

Modeling of Electronic Gates

In this section, the modeling of individual electronic gates by use of GERT will be demonstrated. The following electronic gates will be modeled: AND, OR, NAND (not and), NOR (not or), NOT (invertor) and EXCLUSIVE-OR.

The "AND" gate:

The "AND" gate already has a counterpart in GERT. The logic function of the "AND" gate requires that all inputs must

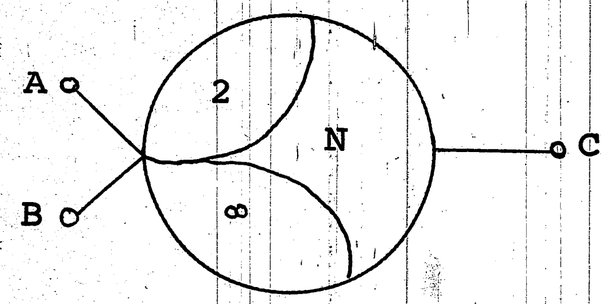
have a value of 1 for the output to equal 1. Otherwise, the output will be 0. (Remember, all output lines of a gate or node will have the same value.) Therefore, a GERT node whose number of releases for realization equals the number of input lines will operate as an "AND" node. (See Figure VII(a)). The number of releases to repeat is infinity. To insure only one realization of the node and consequently accurate modeling of the electronic circuit, the truth function of this GERT node can be easily checked. It complies with the specifications of an "AND" gate. (Figure VII(b)). If there had been four inputs to the node, the gate would look like Figure VII(c). Again the "AND" truth function is satisfied.

The "OR" gate:

The modeling of the "OR" gate, like the "AND" gate, is a simple task. The logic function of the "OR" gate requires that only one input must have a value of 1 to cause the output to equal 1. The output will be 0 only if all the inputs are 0. Therefore, a GERT node, whose number of releases equals 1 will operate as an "OR" node. (Figure VIII(a)). The truth function of this GERT node is equivalent to the truth table of an "OR" gate. This is shown in Figure VIII(b).

GERT Model of "AND" Gate

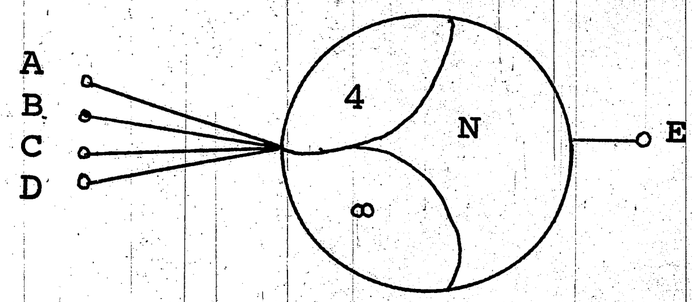
Figure VII.



(a)

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

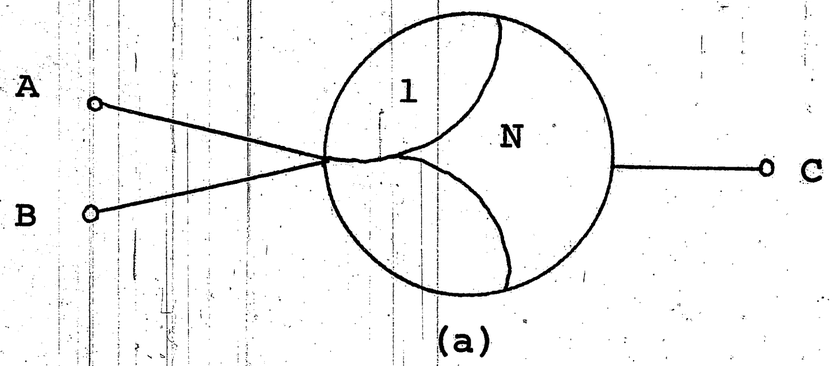
(b)



(c)

"OR" Gate

Figure VIII.



(a)

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

(b)

The clock node:

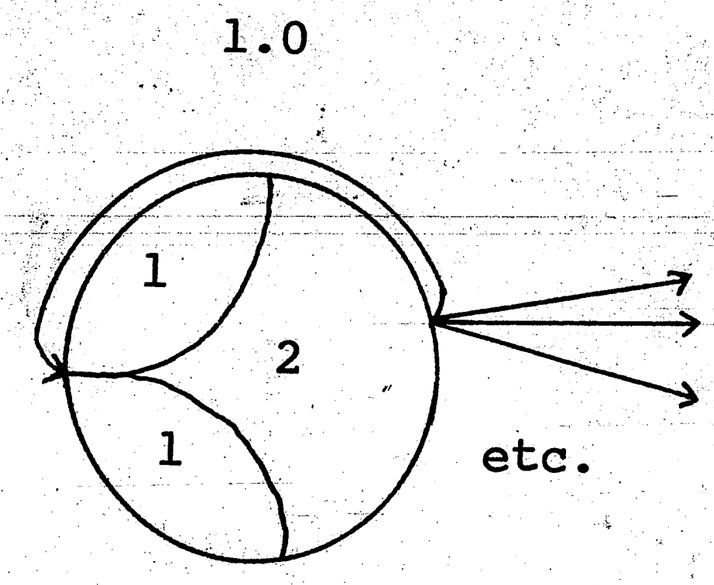
The inversion qualities of the remaining gates to be modeled require the introduction of a clock impulse. This is not the clock implied in the definition of sequential logic circuits. The clock node will generate GERT impulses and transport them to nodes that require such a stimulus. (See Figure IX). The node has an output line that returns to itself (with a delay, in this case, of 1.0 units) to instigate the next clock impulse. The other outputs are connected to the nodes that require the clock. There is no limit on the number of outputs the clock node may have.

The "NAND" gate:

The NAND gate takes the operation of an "AND" gate and then inverts the output. The output value of a "NAND" gate will be the opposite of the output value of an "AND" gate for the same input combination. The truth table of a NAND gate is listed in Figure X. Obviously, this presents a greater difficulty to model than the "AND" and "OR" gates. However, the "NAND" gate has been modeled and it is shown in Figure XI. (The reason this model was chosen will become apparent when we discuss the modeling of the entire circuit.) It can be seen that the clock has been utilized in this design. The node

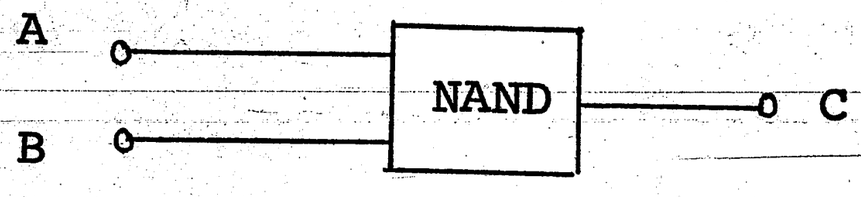
GERT Clock Node

Figure IX.



"NAND" Gate

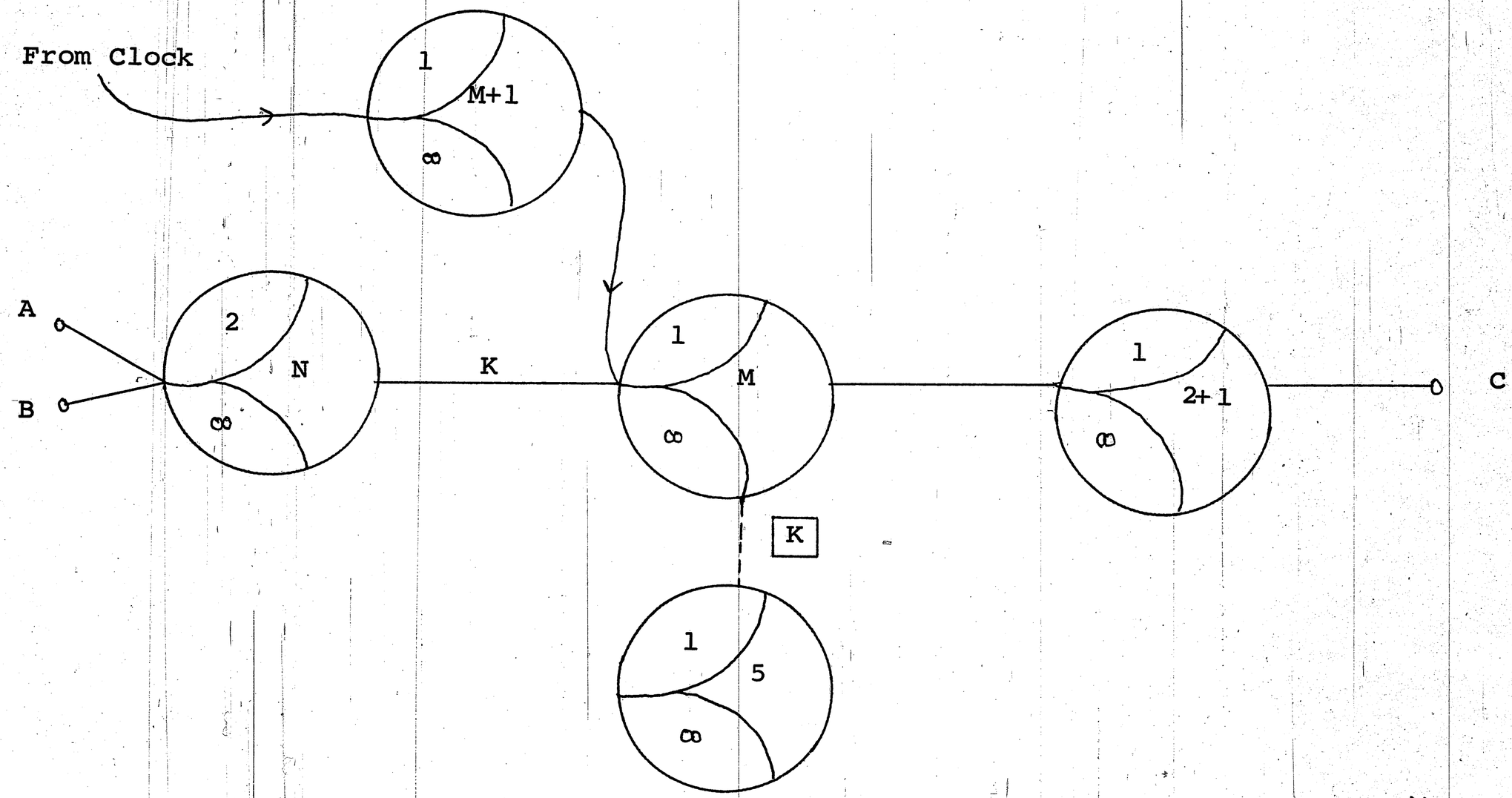
Figure X.



A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

GERT Model of "NAND" Gate

Figure XI.



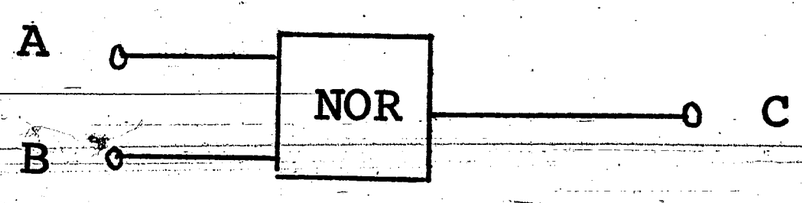
Truth Table of GERT "NAND" Gate

Figure XII.

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

"NOR" Gate

Figure XIII.



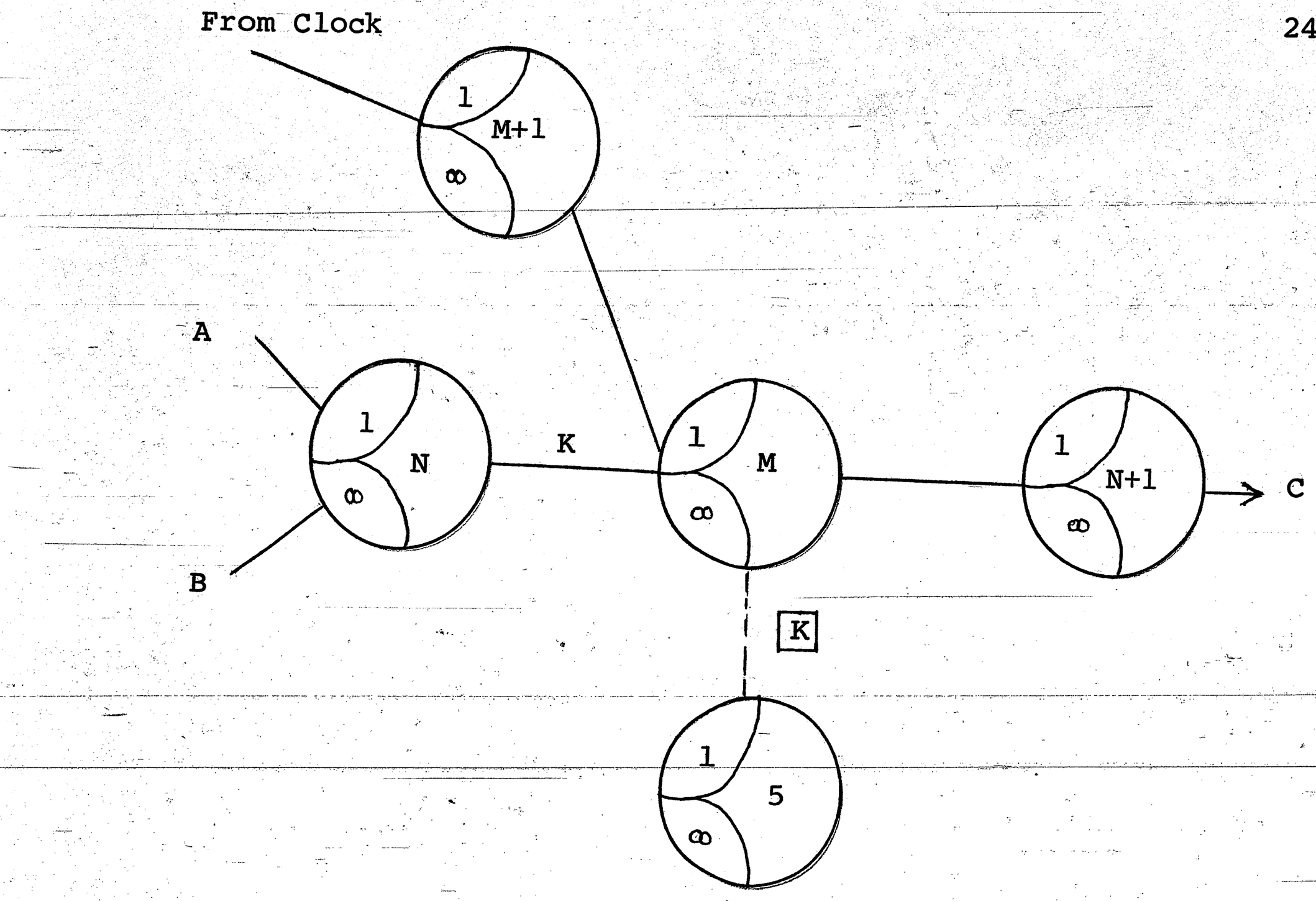
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

modification, the impulse from node M+1 will eventually reach node M and send an impulse to C as desired. The operations described can be put into truth table form. By inspection, it is seen that this table conforms to the operation of a "NAND" gate. (See Figure XII).

The "NOR" gate:

The output of a "NOR" gate is the inversion of the output of an "OR" gate for the same input conditions. The truth table of the "NOR" gate is listed in Figure XIII. The "NOR" gate can be modeled similarly to the "NAND" gate. Again, the node numbers, M and N, and the activity number, k, are a function of the number of the gate that is being modeled. Node 5, as before, is a "sink" node used to kill an unwanted impulse. (See Figure XIV(a)).

The operation of this model is easy to examine. If either A or B have the value 1 the activity k will be activated and the impulse will die in node 5; thereby yielding a zero output. However, if neither A nor B has the value 1, then the clock will strike node M+1 causing it to release; thereby propagating a 1 through the network to the output, C. This can be summarized in a truth table as shown below. This table is



(a)

A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

(b)

GERT Model of "NOR" Gate

Figure XIV.

equivalent to the truth table of a "NOR" gate as desired. (Figure XIV(b)).

The "NOT" gate (invertor):

The "NOT" gate has one input. Its function is to reverse the value of that one input line. That is, if the input is 1 the output must be 0, and vice verse. In truth table form it appears as in Figure XV.

The "NOT" can be thought of as a "NAND" or a "NOR" gate with only one input. The GERT model of a "NOT" gate is precisely that. (Figure XVI(a)).

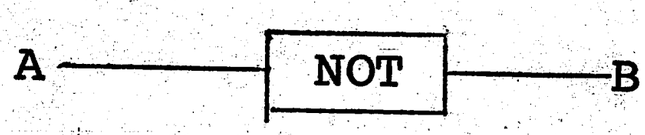
As before, the node numbers, M and N, and the activity number, k, are a function of the number of the gate that is being modeled. The operation of this model is the following: If A has the value 1, the activity k will be realized and the impulse will die in node 5, yielding what is essentially a zero output. If A has the value 0, then the clock impulse will generate a 1 output at B. Figure XVI(b) is the result in table form.

The "EXCLUSIVE-OR" gate:

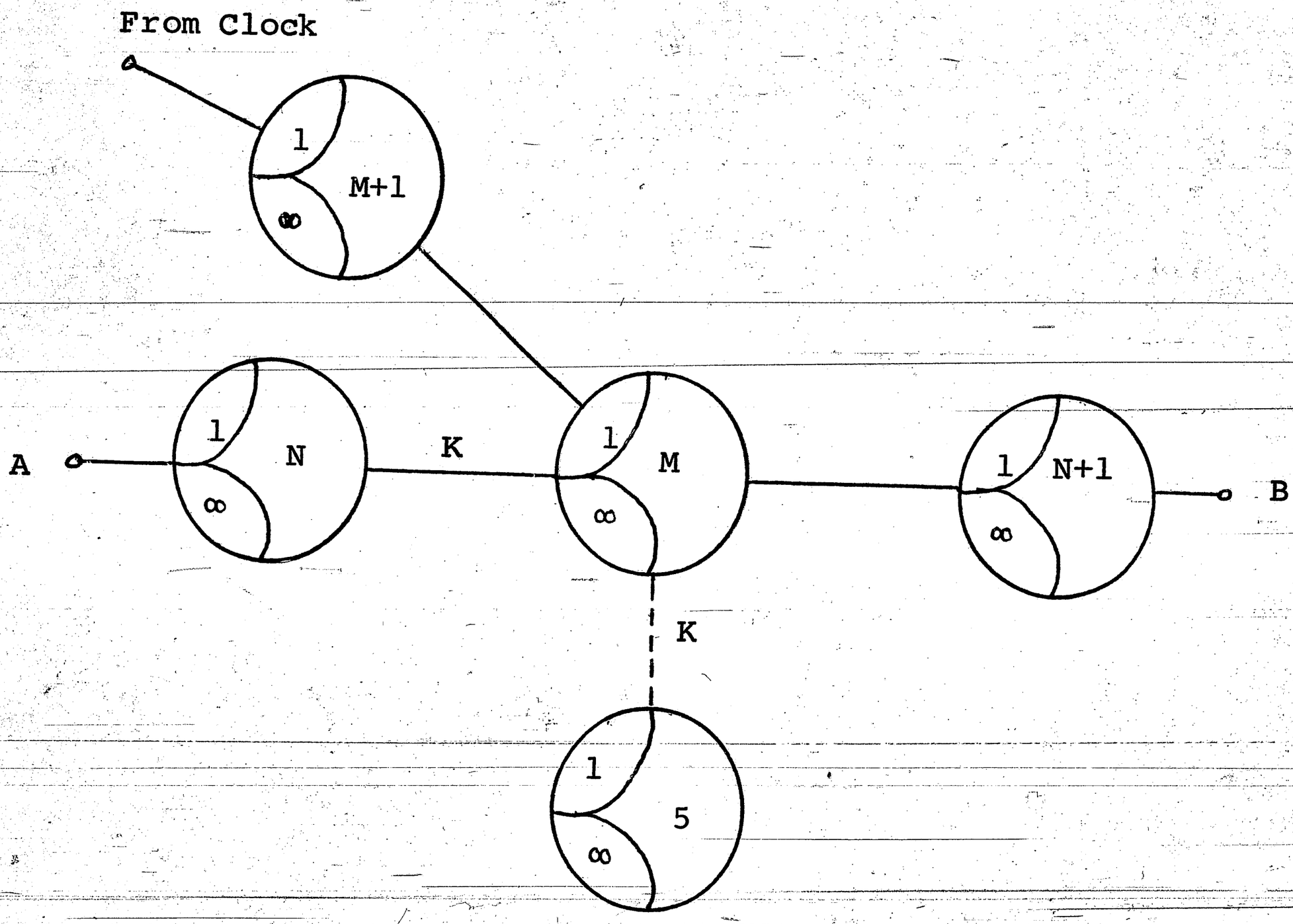
The "EXCLUSIVE-OR" gate is somewhat more difficult to

"NOT" Gate

Figure XV.



A	B
0	1
1	0



(a)

A	B
0	1
1	0

(b)

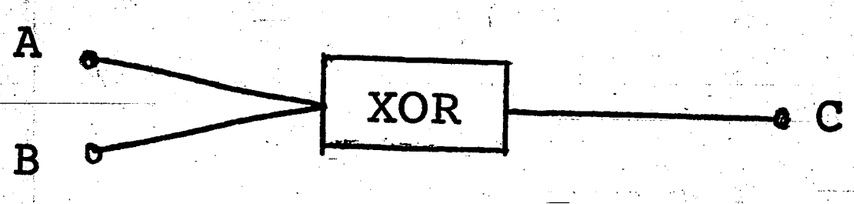
GERT Model of "NOT" Gate

Figure XVI.

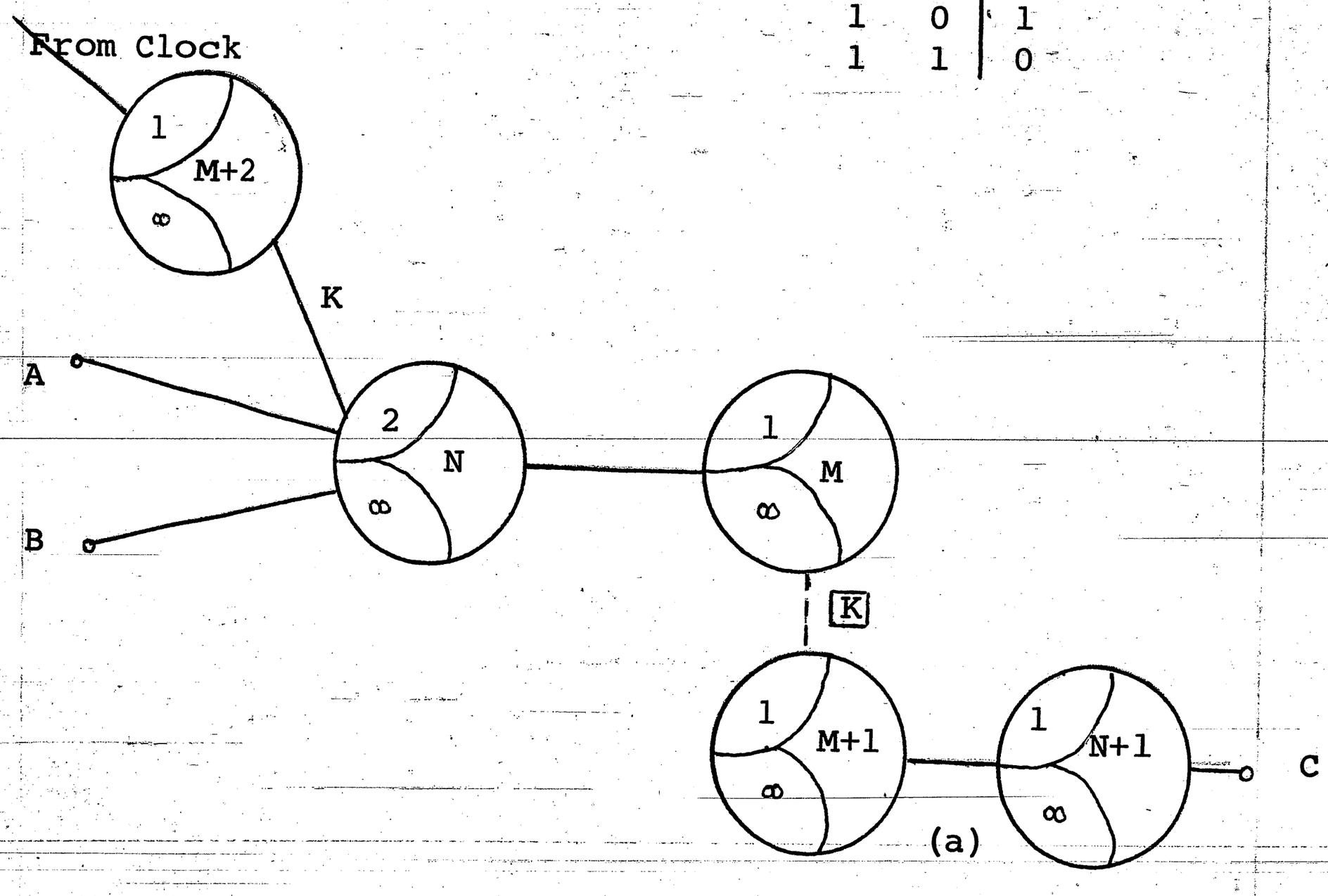
numbers, M and N, and the activity number, k, are a function of the number of the gate that is being modeled. Node 5 is a universal "sink" node to kill an impulse and thereby fulfilling the logic function of the gate. The number of releases on node N must equal the number of input lines to the gate. The clock impulse is "timed" so that the impulse from node M+1 will not reach node M, before an impulse from node N (if there is ever to be an impulse from node N) has reached node M. The "timing" is accomplished by setting the time the clock impulse will strike node M+1 to be the upper bound of the times impulses at A and/or B will strike node N. If node N releases an impulse, activity k will activate the network modification. If that happens, node 5 will assume the position now occupied by node M. The reason for the inclusion of this modification is simple: activity k is activated only if both A and B have the value 1. If that is the case, the desired output at C is 0. Therefore, the impulse from node N must be stopped. This is done essentially by sending it to node 5 which has no output lines. Therefore, the impulse does not reach C. If A and B are not both 1, then activity k will not be realized, and there will be no network modification. Without the

"XOR" Gate

Figure XVII.



A	B	C
0	0	0
0	1	1
1	0	1
1	1	0



GERT Model of "XOR" Gate

Figure XVIII.

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

(b)

model. The "EXCLUSIVE-OR" gate has the distinctive characteristic that it will generate an output of 1, if one and only one of its inputs has the value 1. This characteristic is made clear by truth table notation in Figure XVII.

The model of the "EXCLUSIVE-OR" gate is shown in Figure XVIII(a). Again the values of N, M, and k are a function of the number of the gate being modeled.

If neither A nor B is 1, node N will never be released, yielding a zero output. If both A and B are 1, node N will be released before activity k is realized. Therefore, the line value of 1 transmitted from node N will die in node M. Again, there will be a zero output. If either A or B is 1, but not both, node N will be released only after node M+2 has been struck by the clock. In that case, activity k will have been realized and the modification will have been accomplished. Therefore, the impulse transmitted from node N will go to node M+1, then to node N+1 and to output C. This can be summarized in truth table form. The truth table is equivalent to the truth table of an electronic "EXCLUSIVE-OR" gate. (Figure XVIII(b)).

Complementary Nodes

In addition to the modeling of electronic gates, three other nodes must be created to enable the modeling of electronic circuits by GERT.

The input node:

The input positions in an electronic network can be represented by GERT start nodes. A start node will generate one impulse if it is properly designated. It is represented by the standard GERT notation as shown below. Of course, there are no branches leading into an input node.

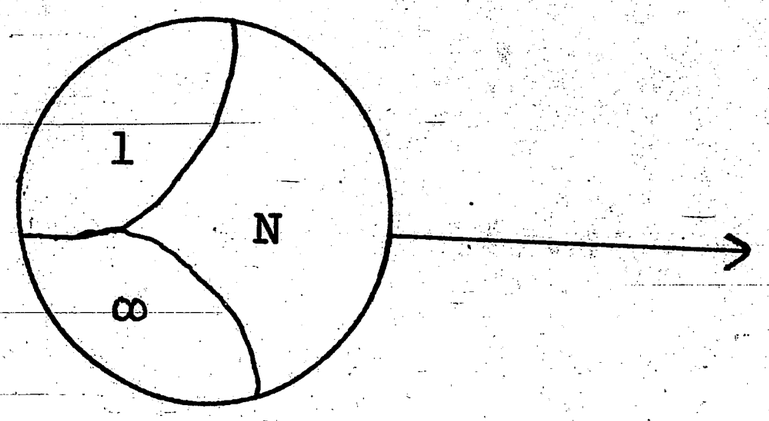
If the input in the electronic network is to have the logical value of 1, the corresponding GERT input node will be designated a start node. If the input is to be 0, the GERT input node will not be designated a start node and of course no impulse will be generated from it. (Figure XIX).

The output node:

The output positions in an electronic network can be represented by a simple GERT node without any output branches. An output node will be placed at each output position in an electronic network. (Figure XX).

Input Node

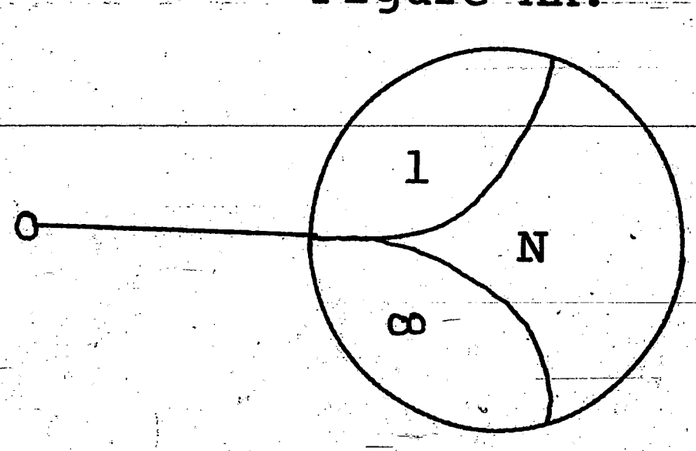
Figure XIX.



Input Node

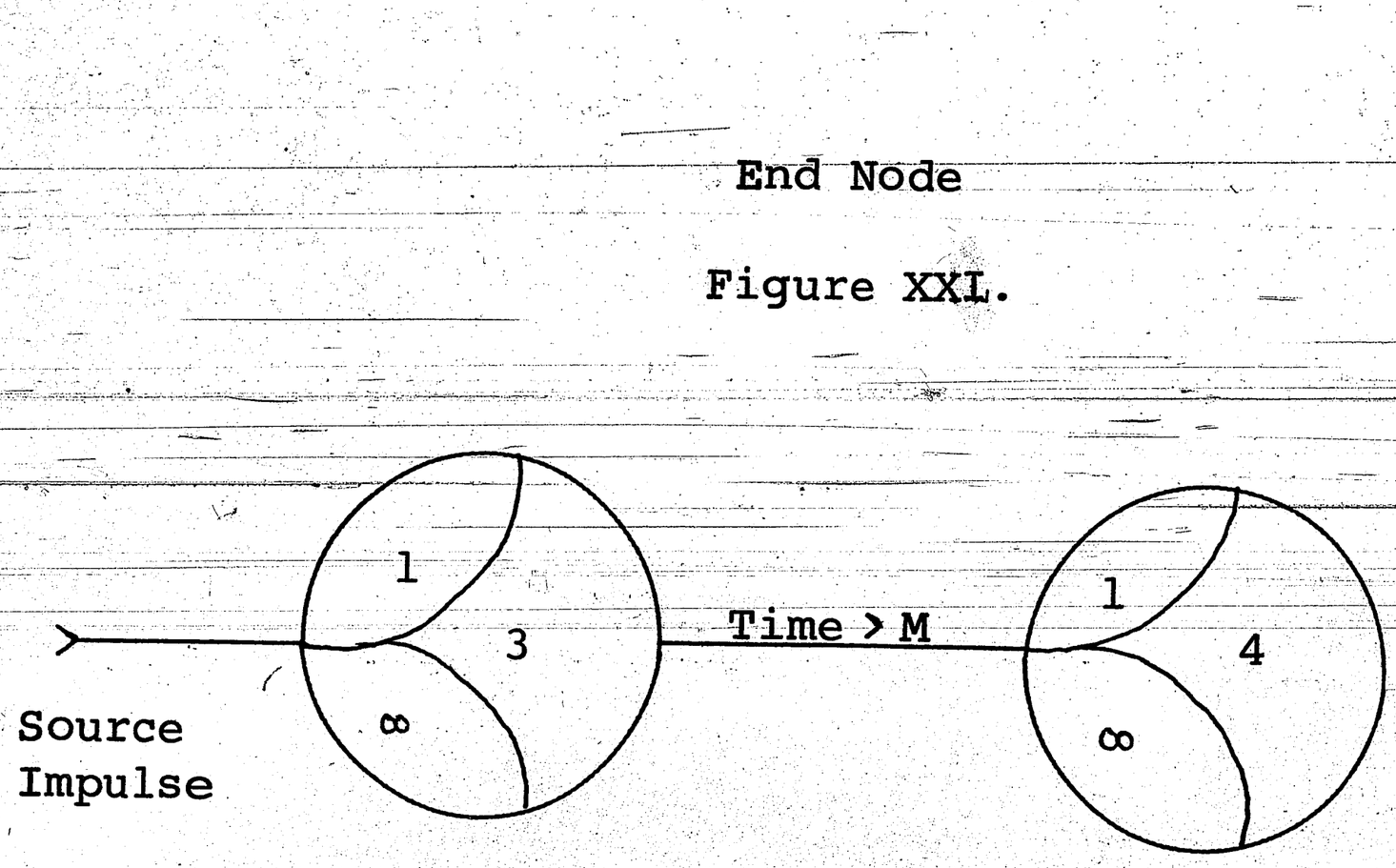
Output Node

Figure XX.



End Node

Figure XXI.



The end node:

The GERTS II Simulator stops processing a network when the proper number of sink nodes have been realized. The nodes which are sink nodes and the required number of sink nodes that must be realized to stop the program are predetermined by the programmer.

One sink node will be used in the modeling of electronic networks. The end node will consist of two nodes, the first of which will be connected to a source impulse. The second node will be the sink node. The two nodes will be connected by a "timed" branch. The "time" that the impulse will take to reach the sink node (and consequently stop the program) must be set sufficiently large to be sure there is no possibility that the program will be "shut-off" before the network has completely run its course. The model of this node is shown in Figure XXI.

Modeling a Combinational Logic Circuit

All elements of a combinational logical circuit have been modeled in GERT notation. In addition to the gate models, complementary nodes have been created to aid in the analysis

of the network. These are the input and output nodes, the "clock" node, and the end node. It is now a simple task to model an entire combinational logic circuit.

The modeling of an electronic network is accomplished by using the following procedure:

(1) Gates

Replace each electronic gate with the proper GERT model. All directed branches remain unchanged.

(2) Input and Output

Place an input node at every input position in the network. Place an output node at each output position in the network.

(3) Clock Impulse

Establish the "clock" node. Connect the "clock" node to all nodes requiring the clock impulse.

(4) End Node

Add the end node and connect it, with the proper time parameter, to the "clock" node.

Steps 3 and 4 can be automatically performed by the computer program.

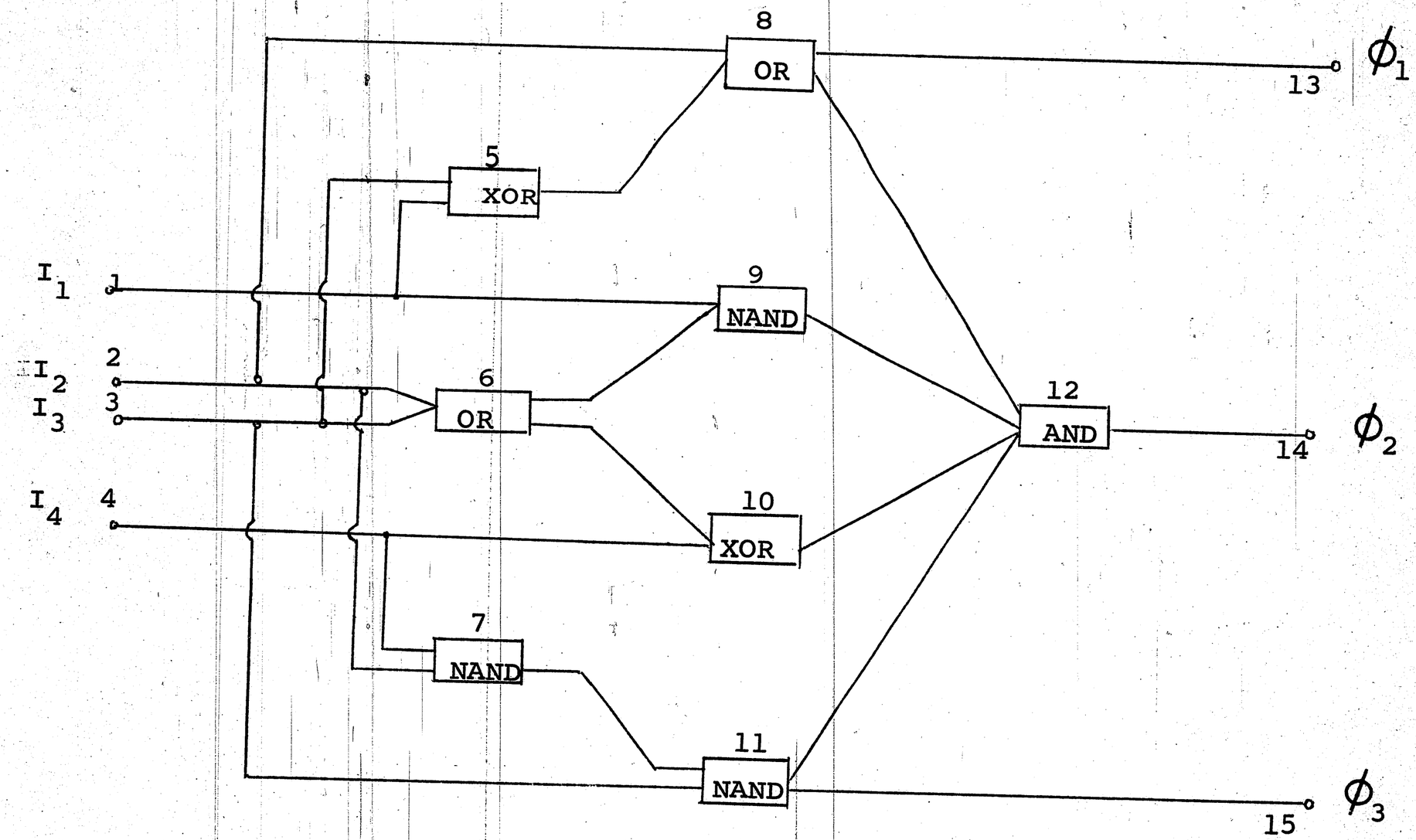
To illustrate the modeling of combinational logic network, an example of a circuit is shown in Figure XXII and its equivalent is shown in Figure XXIII.

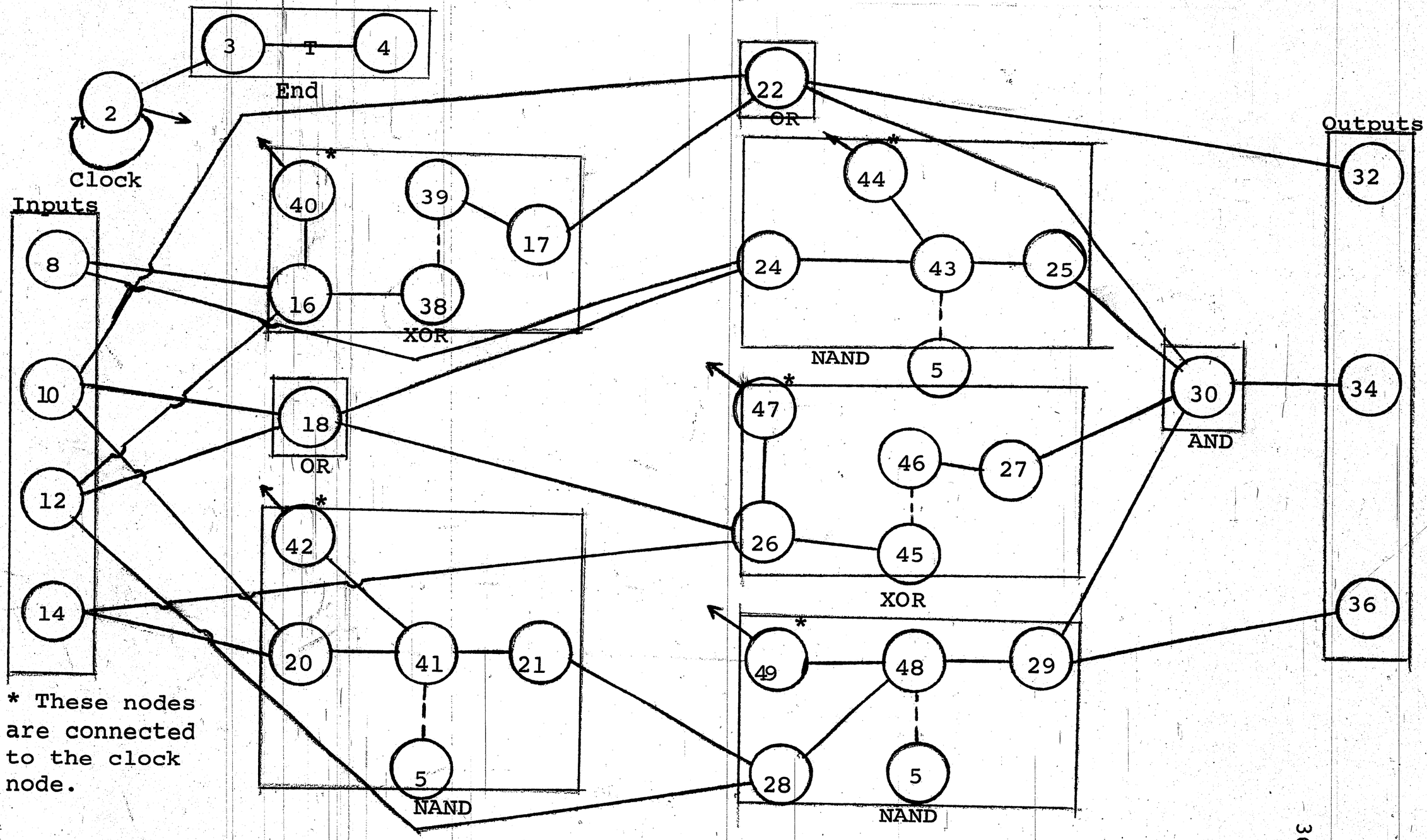
All gates except the "NOR" gate are present in this network. Before the transformation to GERT notation is started, it is convenient to number each gate and each branch in a consecutive fashion. The gates should be numbered such that the number of any gate will be larger than the number of any gate that is preceded by it. For example, the gate number of the AND gate must be higher than the numbers of the gates that feed into it. The input and output positions should also be numbered as if they were gates. The numbering has already been placed on the diagram.

The equivalent GERT network is shown on the preceding page. The number of releases for each node has been deleted for clarity. The nodes have been grouped and titled to demonstrate the relationship between the electronic network and the GERT network. Also, the connecting branches between

Combinational Logic Circuit

Figure XXII.





* These nodes are connected to the clock node.

GERT Model of Figure XXII.

Figure XXIII.

the clock node and the nodes requiring the clock impulse have been eliminated to keep the diagram from becoming over-cluttered. The node numbers are the numbers the computer simulation would give them if this network was analyzed.

Sequential Logic Circuits

The modeling of sequential networks is a more difficult problem than that of combinational circuits. Sequential circuits differ from combinational logic circuits in that the present state of the circuit has an effect on the next state of the circuit, whereas in a combinational logic circuit, the next state is independent of the present state. A record of the "past history" of the circuit is made possible by "memory" elements and feedback paths. The basic memory elements are flip-flops. Each flip-flop stores one bit of information. Thus, the information storing capacity of a circuit is dependent on the number of flip-flops it contains. Another new element encountered in sequential circuits is the circuit clock. The general concept of the clock is that it generates an impulse at periodic intervals. In performing this stroboscopic activity it is instructing the circuit to evaluate the inputs

at these and only these specified times. This will be explained further when the modeling of flip-flops is considered.

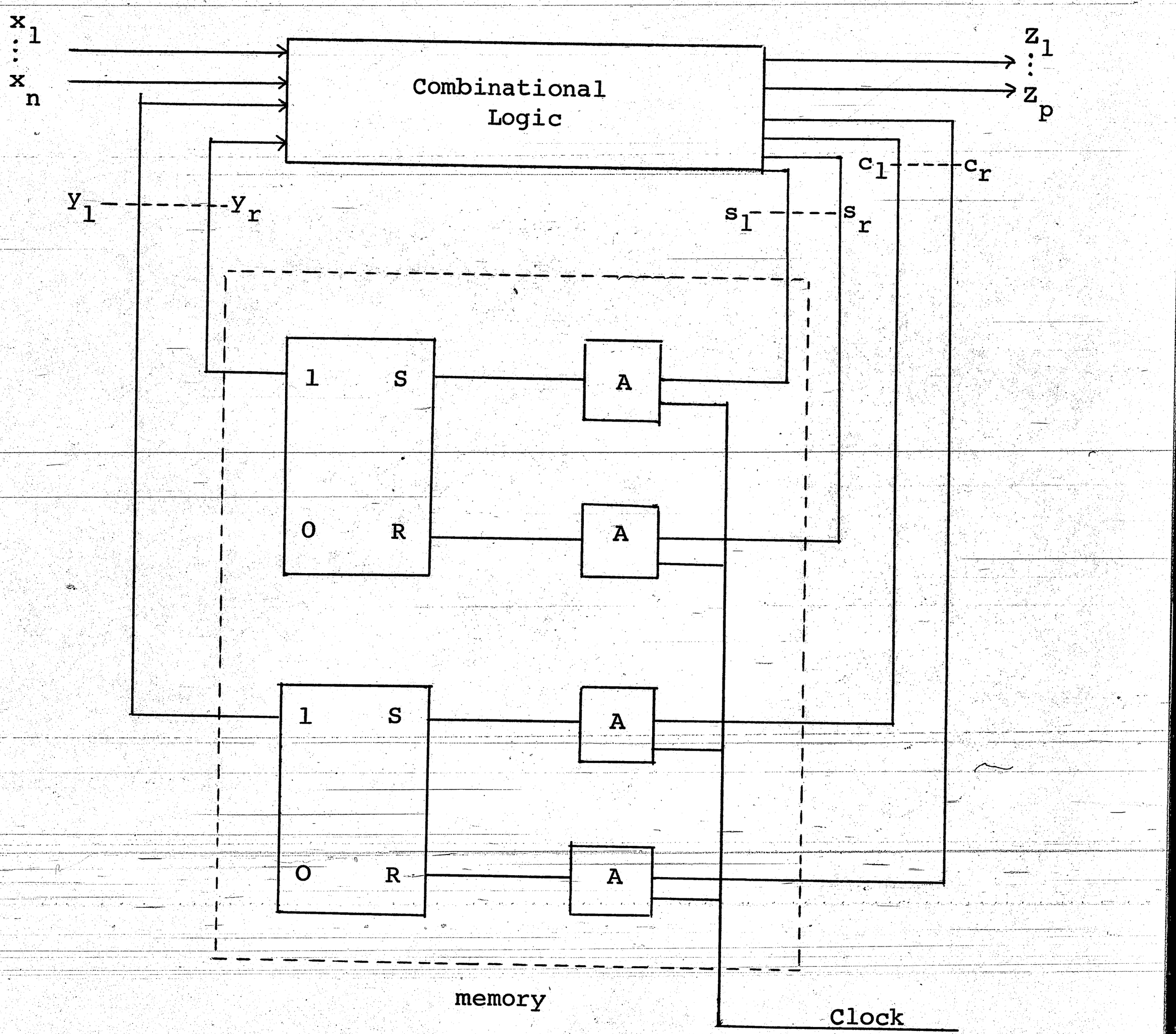
The general model of clocked sequential circuits is shown in Figure XXIV. Thus, the problem of modeling sequential circuits can be clearly defined. It involves the graphical portrayal of the separate parts, combinational logic and memory units, and the delicate and exacting integration of them. Timing is an essential part of the design of sequential circuits and will be the major problem involved in modeling sequential circuits. The development of a simple to use, easy to adapt computer method of analysis of sequential circuits would not only be useful in itself but also would provide a tool for searching and testing new fault detection algorithms.

Modeling Flip-Flops

There are three major types of flip-flops: toggle, set-reset, and j-k flip-flops. They are different in their specifics but similar in concept. There are two output lines eminating from a flip-flop. One line is labeled the "1" output, the other "0". At all times, one output line is at the

General Model of Clocked Sequential Circuit

Figure XXIV.

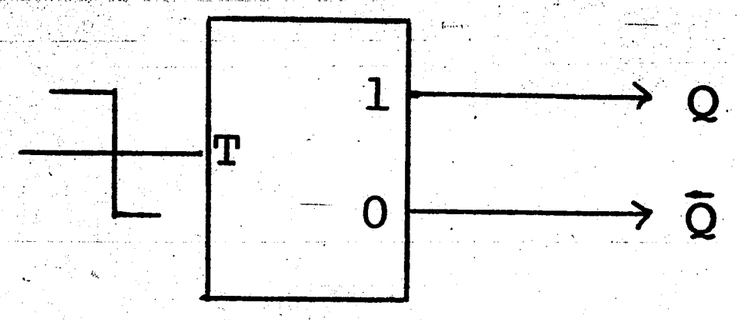


electronic value of 1 and the other at 0. That is, if the "1" output has the electronic value 1, the "0" output is necessarily 0, and vice versa. The state of the flip-flop is defined as the value of the output labeled "1". Therefore, it is said that if the "1" output is on (equal to 1) the flip-flop is in state 1, and if the "0" output is on the flip-flop is in state 0.

The state of the flip-flop at any given time depends not only on the inputs to the flip-flop but also its present state. The truth table formulations of the three types of flip-flops are shown in Figure XXV. The concept of "present state" and "next state" is probably the most important notion to grasp. In this application, it will be assumed that the clock width and interval are such that asynchronous operations do not occur. That is, as soon as the flip-flop senses new imputs, it begins to change its state. This transition is not completed until the input phase has been completed. However, the transition is completed before the next combination of inputs arrive. Thus the present state of the flip-flop is defined by its output just before and during the registration of new inputs, and

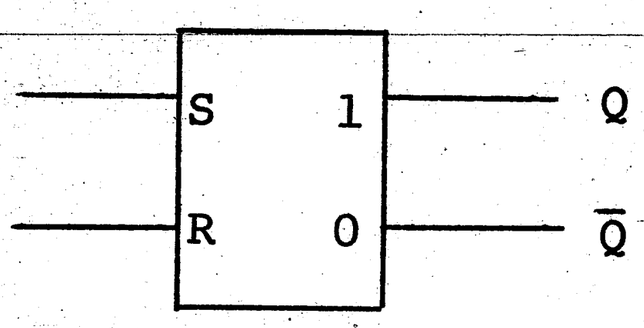
Truth Tables of Flip-Flops

Figure XXV.



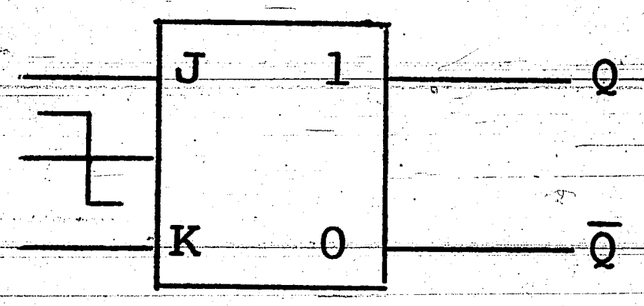
Before Trigger Pulse		After Trigger Pulse	
Q	$\bar{Q}$	Q	$\bar{Q}$
0	1	1	0
1	0	0	1

Toggle Flip-Flop



Input		Output	
S	R	Q	$\bar{Q}$
0	0	no change	
1	0	1	0
0	1	0	1
1	1	undefined	

Set-Reset Flip-Flop



Input		Output	
J	K	Q	$\bar{Q}$
0	0	no change	
0	1	0	1
1	0	1	0
1	1	compliment	

J-K Flip-Flop

the next state is that state which the flip-flop exhibits after the inputs have propogated through the circuit. Thus, the state changes are linked to the clock. If the clock width was not as assumed (i.e., if it was a wide clock) the next state might be generated before the present inputs have fully disseminated, therefore, causing a potentially unstable situation. (In some applications, the wide clock is desirable. This thesis will first deal with the regular clocking activity and will later show that the wide clock does not pose any insurmountable difficulties.)

Thus, the present state of the flip-flop yields the values it imposes upon the circuit at the time of the present set of inputs, and the next state (which is determined in the present) establishes the values the flip-flop will impose on the circuit at the next set of inputs. Looking at the operation of a toogle flip-flop, whose initial state is 0, this concept of present and next state can be clearly shown. When a trigger (synchronized with the clock) reaches the flip-flop it resets the flip-flop to the state 1. However, the flip-flop continued

to generate the state 0 at least until the clock pulse ended. Thus, the state 0 accompanied the present set of inputs (whatever they may be) through the circuit, and the next state, 1, has been made ready for the next set of inputs.

Therefore, the essence of modeling flip-flops is to delay the progression of the next state until the next clock impulse arrives. This is easily accomplished due to the "timing" nature of the directed paths in GERT. Thus, when an impulse causes a flip-flop to change state, the new state is scheduled for the time the new inputs will arrive.

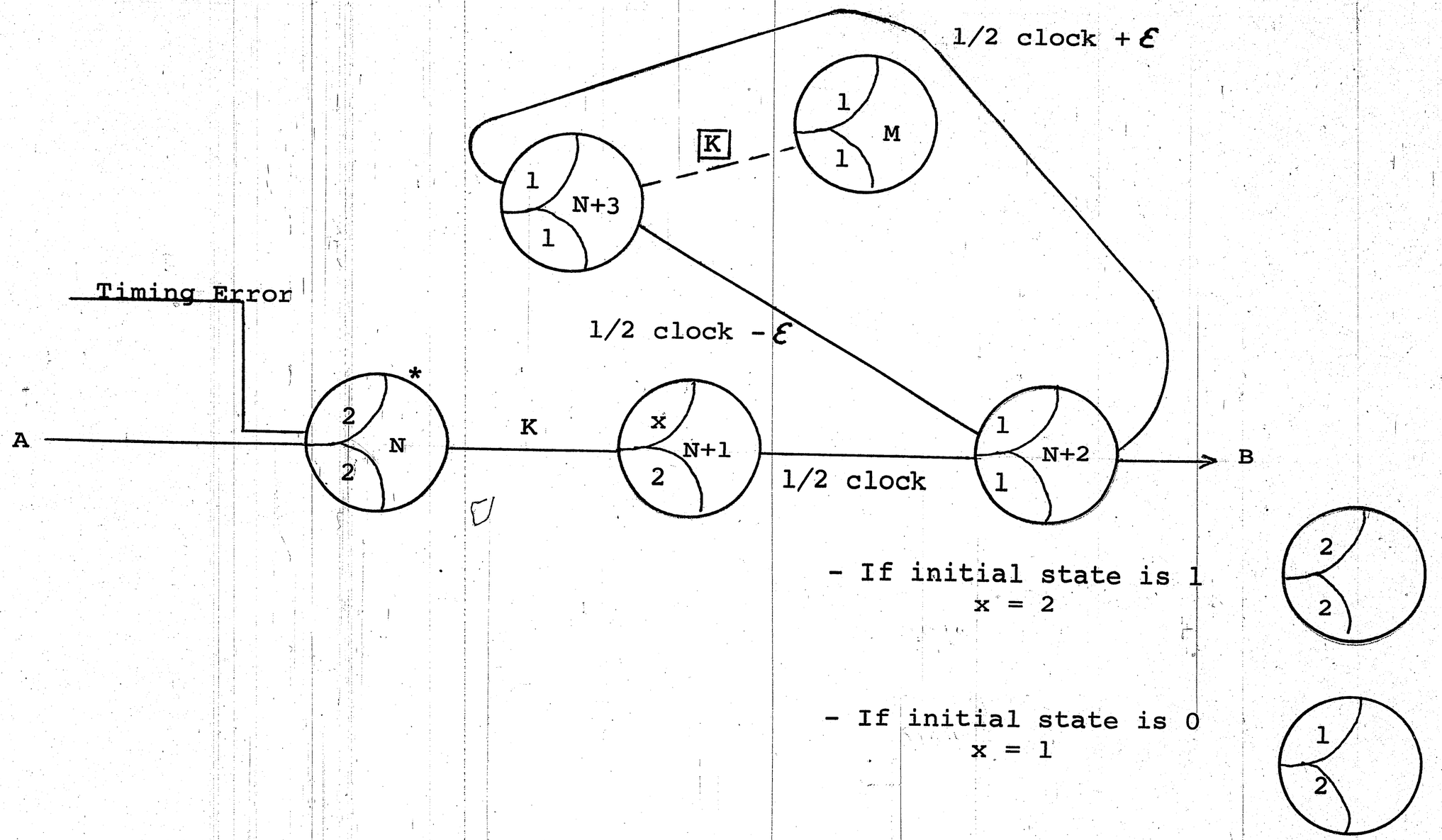
The Toggle Flip-Flop

The discussion in the last section expresses in part the concept behind the toggle flip-flop. The flip-flop will change state with the registration of a trigger. A GERT model of this type of flip-flop is shown in Figure XXVI. For simplicity consider only the "1" output line. (The timing error has much the same function it had in combinational circuits and is generated by the GERT clock.) If a trigger reaches node N, it will cause an impulse to strike node N+1. If node N+1 is

released then the output will go to "1" at a time equal to 1/2 clock from the present. The scheduling of this "1" output will have no effect of the present inputs, but instead will accompany the next set of inputs. That is, the "1" is the next state of the flip-flop.

If there is no trigger at the time of new inputs the state of the flip-flop will remain the same. That is, the next state will equal the present state. The present state can be "saved" by directing it at node N+3. If there is no trigger, the last state will eventually reach node N+3 and generate the next state. If there <u>is</u> a trigger as discussed in the last paragraph, the retained last state must be ignored. This is accomplished by activity k. If there is a trigger, node N+1 will be released and modification k will be activated. This will send the last state into node M. Node M has no outputs; hence the last state simply is eliminated.

The asterisks denotes a node that must be reset after each set of inputs are imposed. In this case, the number of releases on node N must be two before the next set of inputs are sent into the circuit. If, for a given set of inputs, there is no

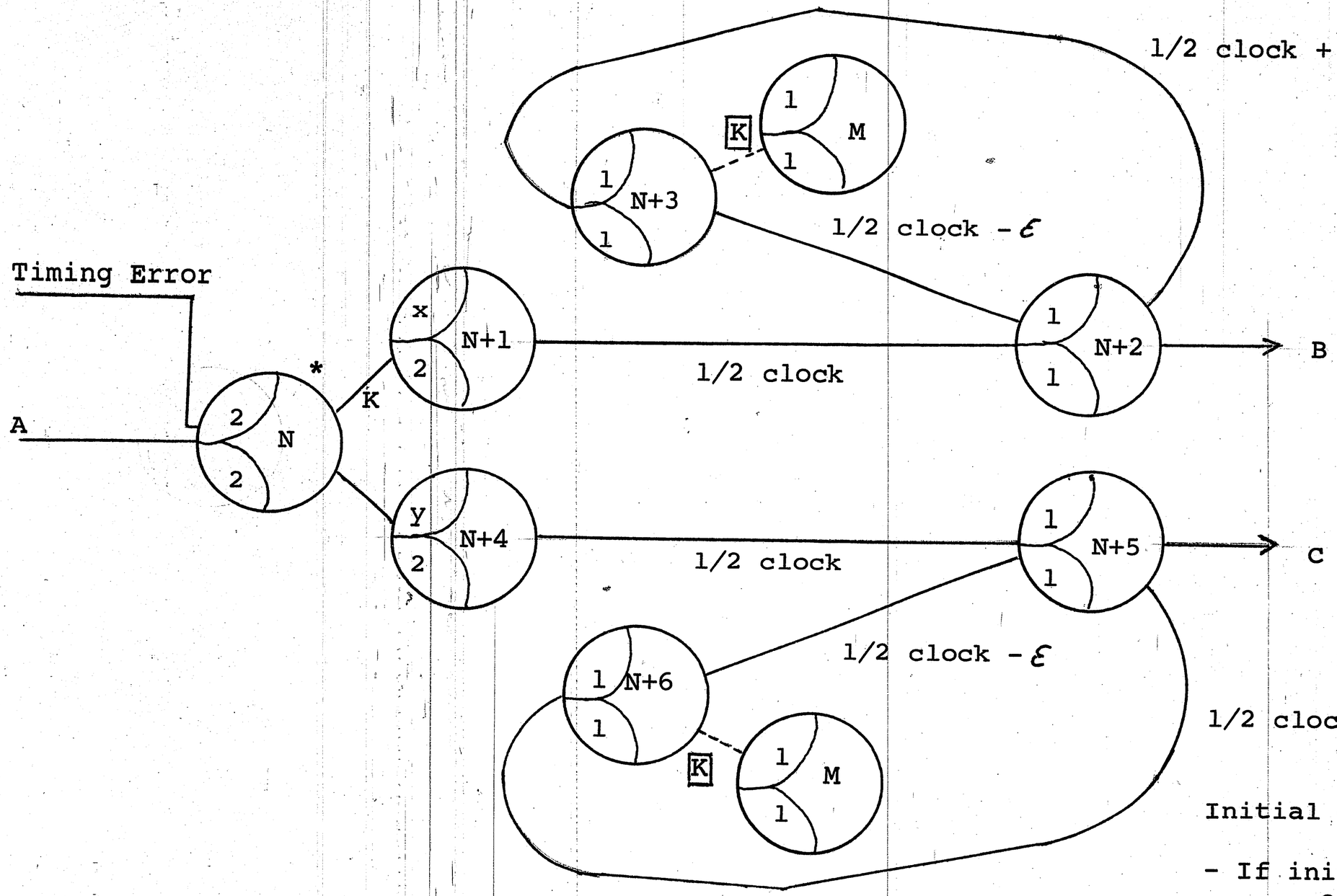


- If initial state is 1
  x = 2

- If initial state is 0
  x = 1

GERT Model of Toggle Flip-Flop
(one output)

Figure XXVI.



GERT Model of Toggle Flip-Flop
(two outputs)

Figure XXVII.

Initial Conditions:

- If initial state is 1
  x=2, y=1

- If initial state is 0
  x=1, y=2

trigger for this flip-flop, node N will be struck once (by the GERT clock). By resetting the node it is made ready for the next set of inputs.

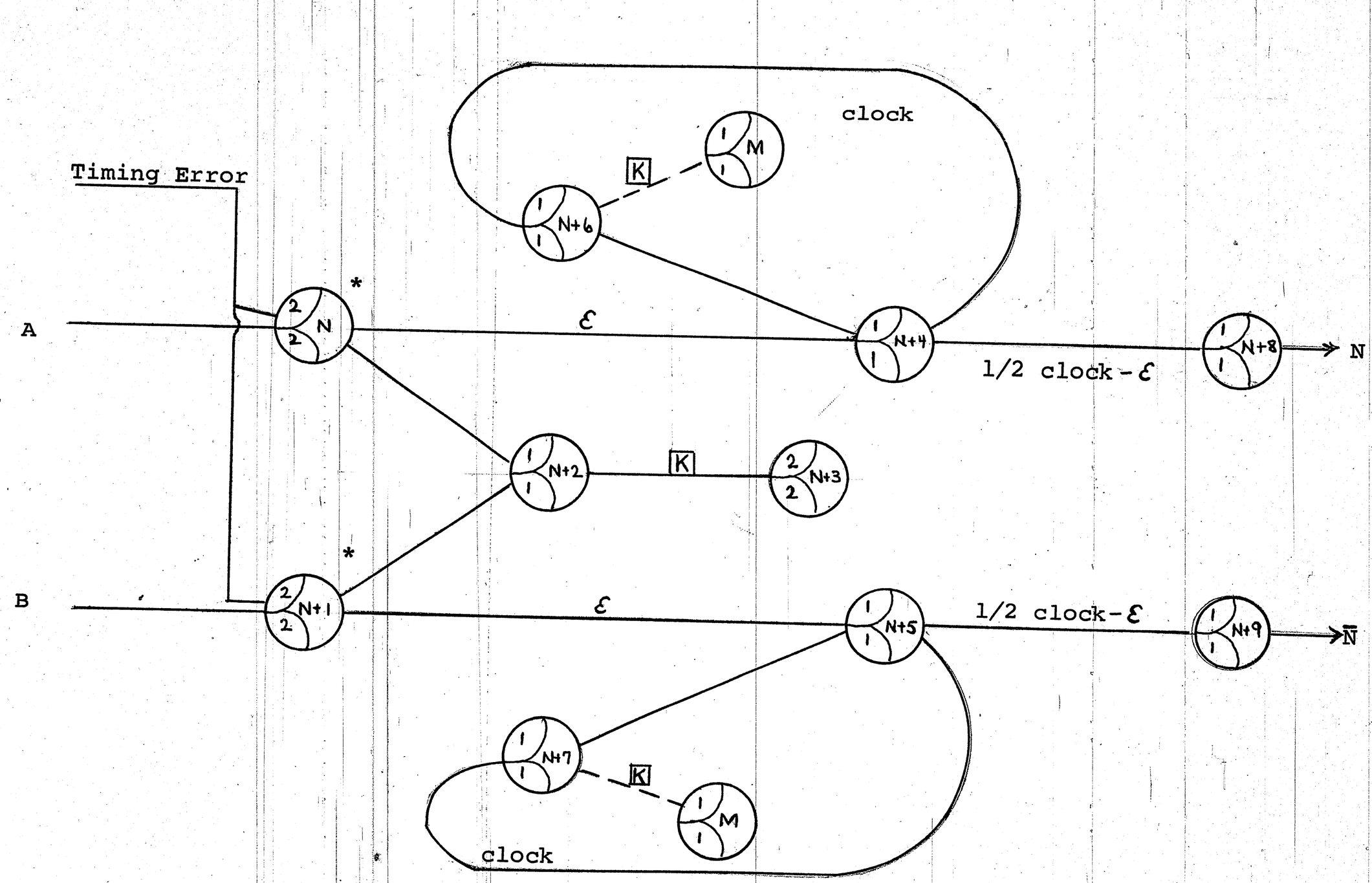
Figure XXVII shows the situation where both outputs are considered. The model is quite similar to Figure XXVI, with an additional storage unit for the 0 output line. The values of X and Y are dependent upon the desired initial state.

Set-Reset Flip-Flop

The set-reset (s-r) flip-flop has two input lines. (See Figure XXVIII). The outputs behave according to the table in Figure XXV. The inputs combination (1, 1) is not allowed to occur. That is, a circuit in which an s-r flip-flop is employed must design out the possibility of a (1, 1) input to the flip-flop.

Again, the timing error is generated by the GERT clock. Its function is basically to synchronize the two inputs, A and B.

From the truth table, it can be seen that the inputs (1, 0) and (0, 1) "pass" unobstructed through the circuit and the



GERT Model of Set-Reset Flip-Flop

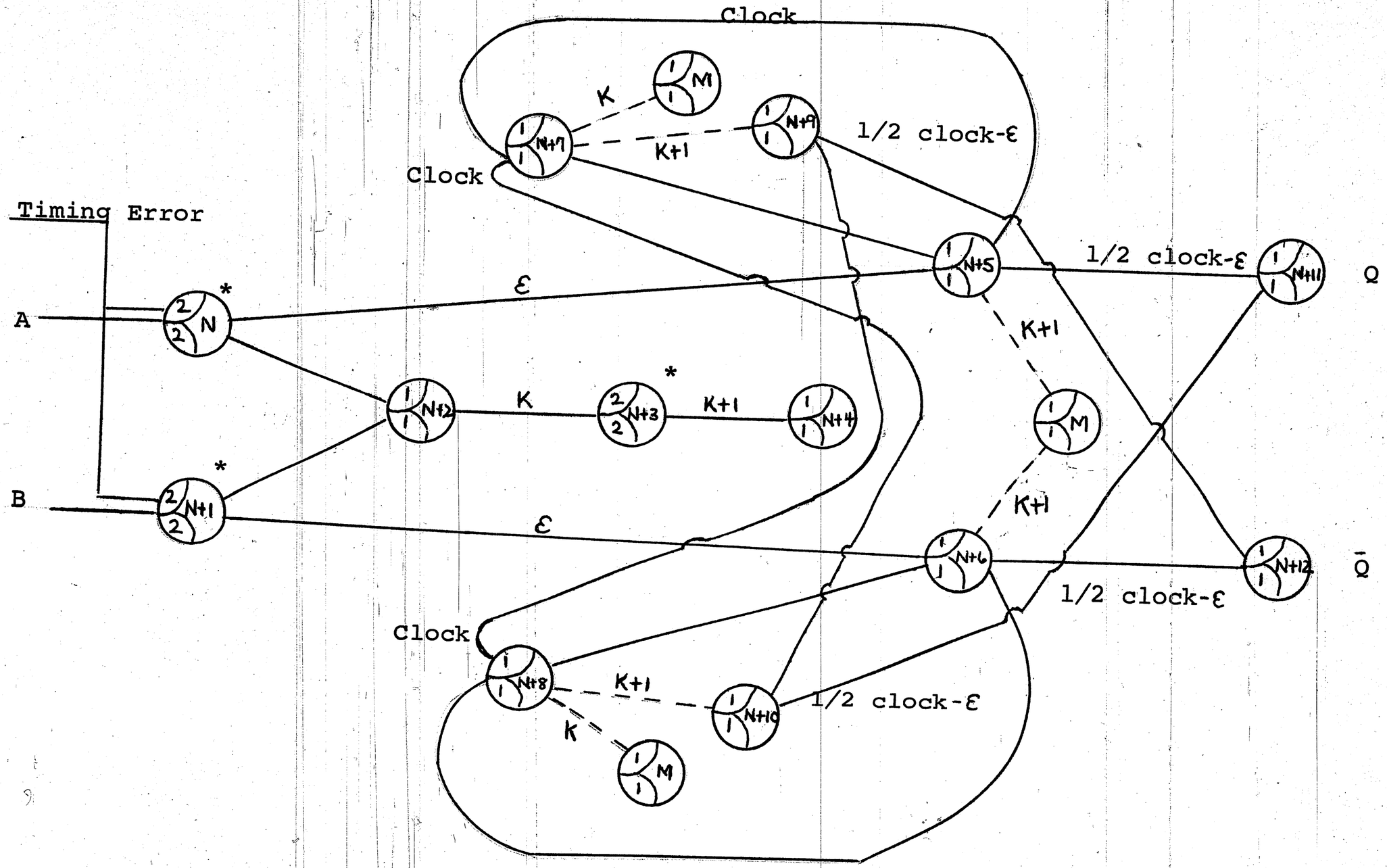
Figure XXVIII.

last state is ignored. In the GERT model, this is accomplished by activity k. If the inputs are (0, 0), the modification is not instituted and the next state is the last state. The astericks denote the nodes that should be reset before the next clock pulse.

The timing of the activities accomplishes the desired relationship between states and inputs. The epsilon time is required to insure the activity k is afforded the opportunity to be effective. The epsilon is later substracted out so that there is no build-up of error. The value of 1/2 clock-epsilon must be greater than the time of reset; that is, the time between the initiation of the clock pulse and the time the resetting of nodes occur. Otherwise, the next state could be lost in the resetting.

The j-k Flip-Flop

The j-k flip-flop is modeled in Figure XXIX. It is essentially the same as the s-r flip-flop; however, the (1, 1) input contingency has been accounted for. In that case, the next state is the complement of the present state. Referring



GERT Model of J-K Flip-Flop

Figure XXIX.

to Figure XXIX, if (1, 1) is recorded at (A, B), node N+4 will be realized, thus setting activity k+1 into action. This modification accomplishes two things: (1) it sends the inputs into "sink" node M, and (2) it switches the memory functions thus yielding a next state which is the complement of the present state. For other input combinations, the network performs similarly to the s-r flip-flop.

There are 14 nodes in this representation of the j-k flip-flop. However, due to the fact that only one output ever has the value "1", far fewer activities are realized than might appear.

Modeling a Sequential Circuit

The basic elements of a sequential logic circuit have now been developed with the specific intention of modeling complex circuits. That is, the flip-flop prototypes explained in the previous section have the inherent characteristics enabling the simulation of an entire circuit simply by joining the GERT representations of its individual components. Modeling of the flip-flop proved to be serindipitously salutary with respect to

the larger problem, the exacting integration of combinational and sequential elements. That is by seeking graphical paragons of individual flip-flops, the previously elusive nexus required to model entire circuits was uncovered. Therefore, it is now possible to consider the modeling of an entire circuit.

The procedure is very similar to the technique outlined when combination circuits were discussed:

(1) Replace each electronic gate with the proper GERT model.

(2) Add input and output nodes.

(3) Establish both the GERT clock and the circuit clock. (In actuality, these two can be one in the same. But for purposes of illustration, the distinction will be made here.)

(4) Add an end node.

(5) Assign appropriate "times" to directed branches.

(6) Create inputs as determined by the programmer.

(7) Establish reset nodes. The network modifications caused by one set of inputs must be returned to their original state. Likewise, nodes with multiple releases must be reset. There can be one reset for combinational elements and one for sequential or, as in the case of the clock, they can be the same.

(8) Set flip-flops to desired initial state.

Most of what is listed can be accomplished by the computer program.

To demonstrate this procedure, three circuits have been modeled and their proper operation has been modeled. A test of the j-k flip-flop is not found in the other trials.

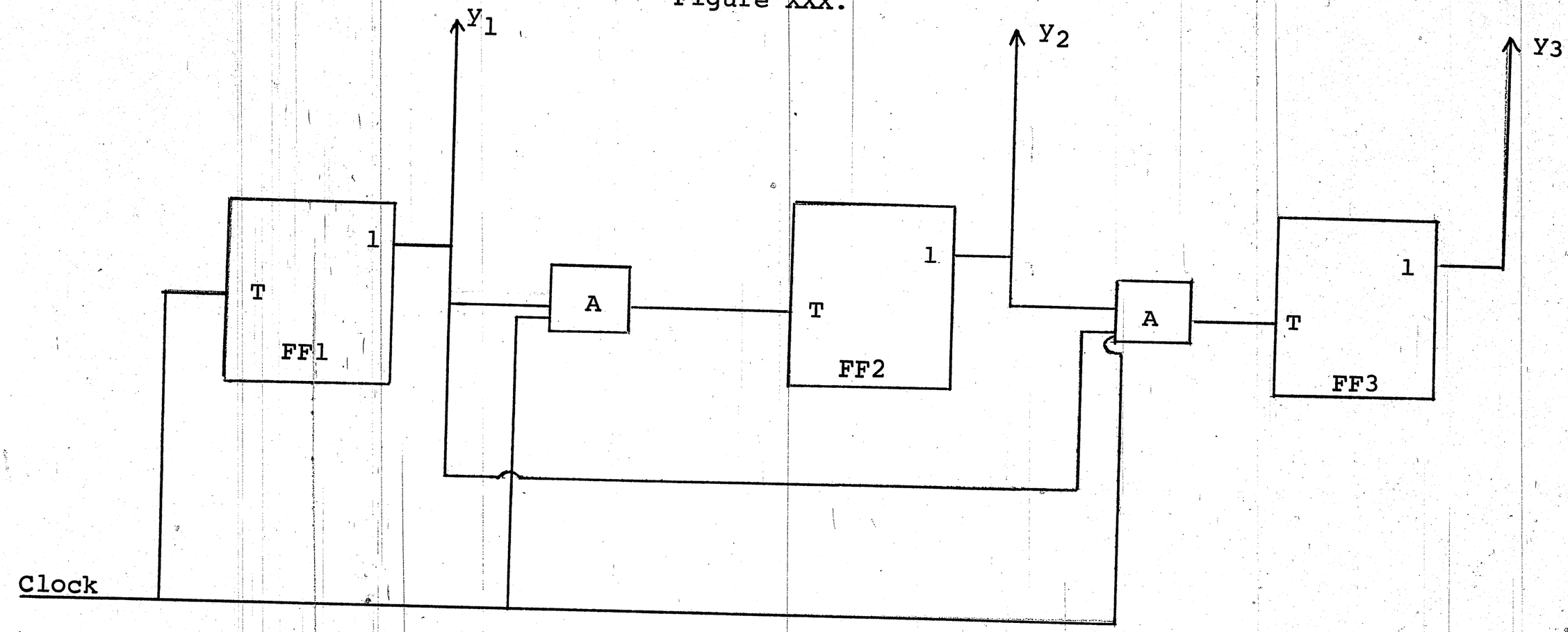
Modulo "8" Counter

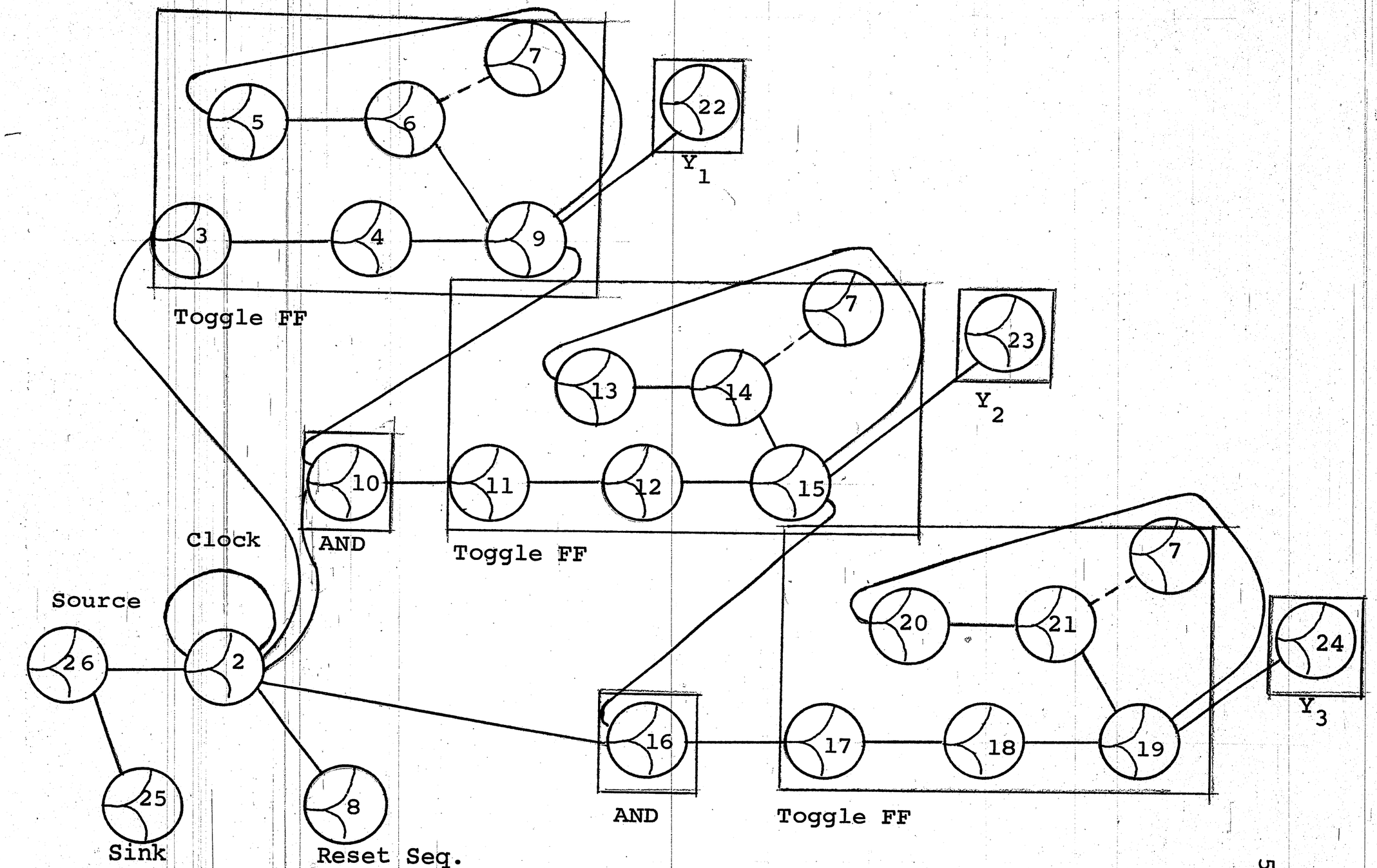
Digital computers use counters in many ways. One way is an instruction counter. That is, when an instruction is executed, the counter is incremented. This directs the computer to the next instruction.

A counter of this type is shown in Figure XXX. It has three toggle flip-flops. Initially all are at state "0". The

Modulo "8" Counter

Figure XXX.





GERT Model of Modulo "8" Counter

Figure XXXI.

first pulse sets the first flip-flop to 1. The next pulse will shift this bit into flip-flop 2 and reset flip-flop 1 to 0. The activity is obviously that of a binary counter.

The GERT model of this circuit is shown in Figure XXXI. Due to the simplicity of the circuit, the timing error activities have been deleted. The GERT network has been programmed and run. The results match the desired outcome. See Appendix A.

Shift Register

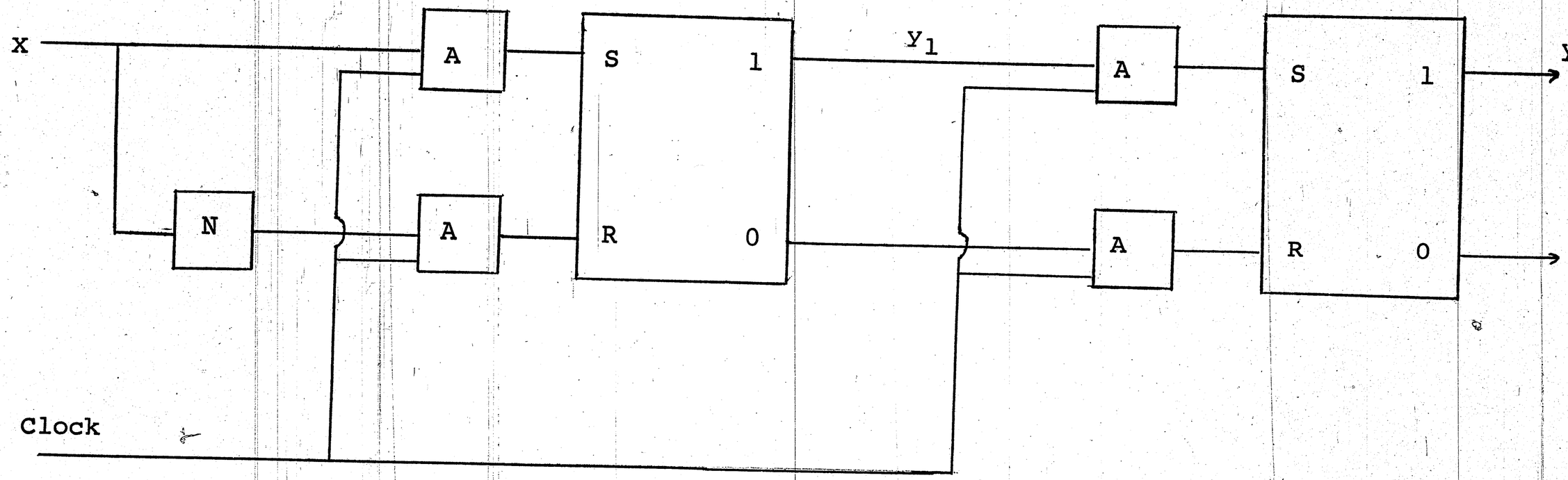
A counter that is used to store related bits of information is known as a register. If the information is serialized by its source, the counter can be called a shift register.

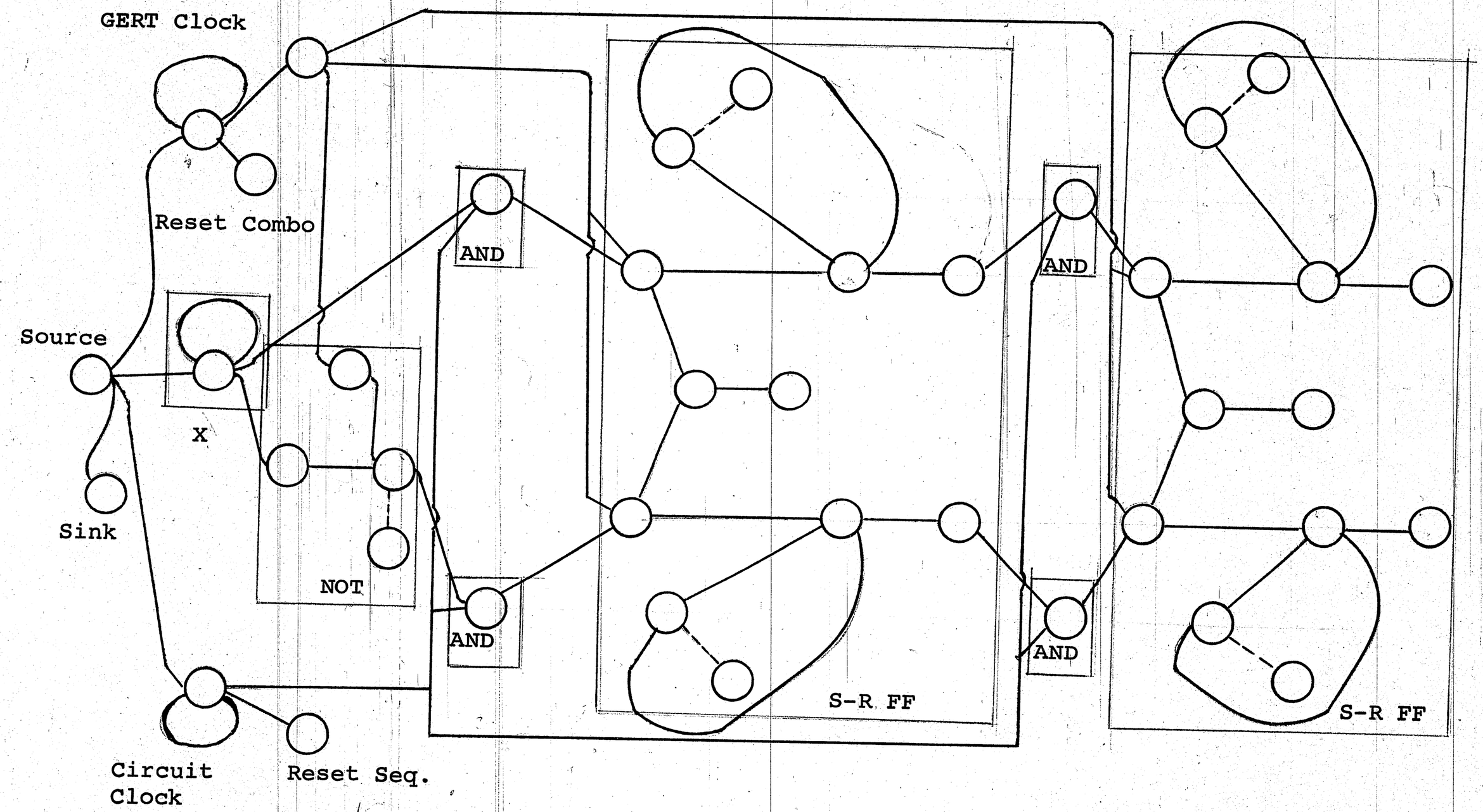
Figure XXXII is an illustration of a shift register with two set-reset flip-flops. Figure XXXIII is the GERT representation of the shift register. In this case, the flip-flops are both initialized to state 0.

This circuit has been programmed and run on the computer. The results are listed in Appendix B and are completely satisfactory.

Shift Register

Figure XXXII.





GERT Model of Shift Register

Figure XXXIII.

Test of j-k Flip-Flop

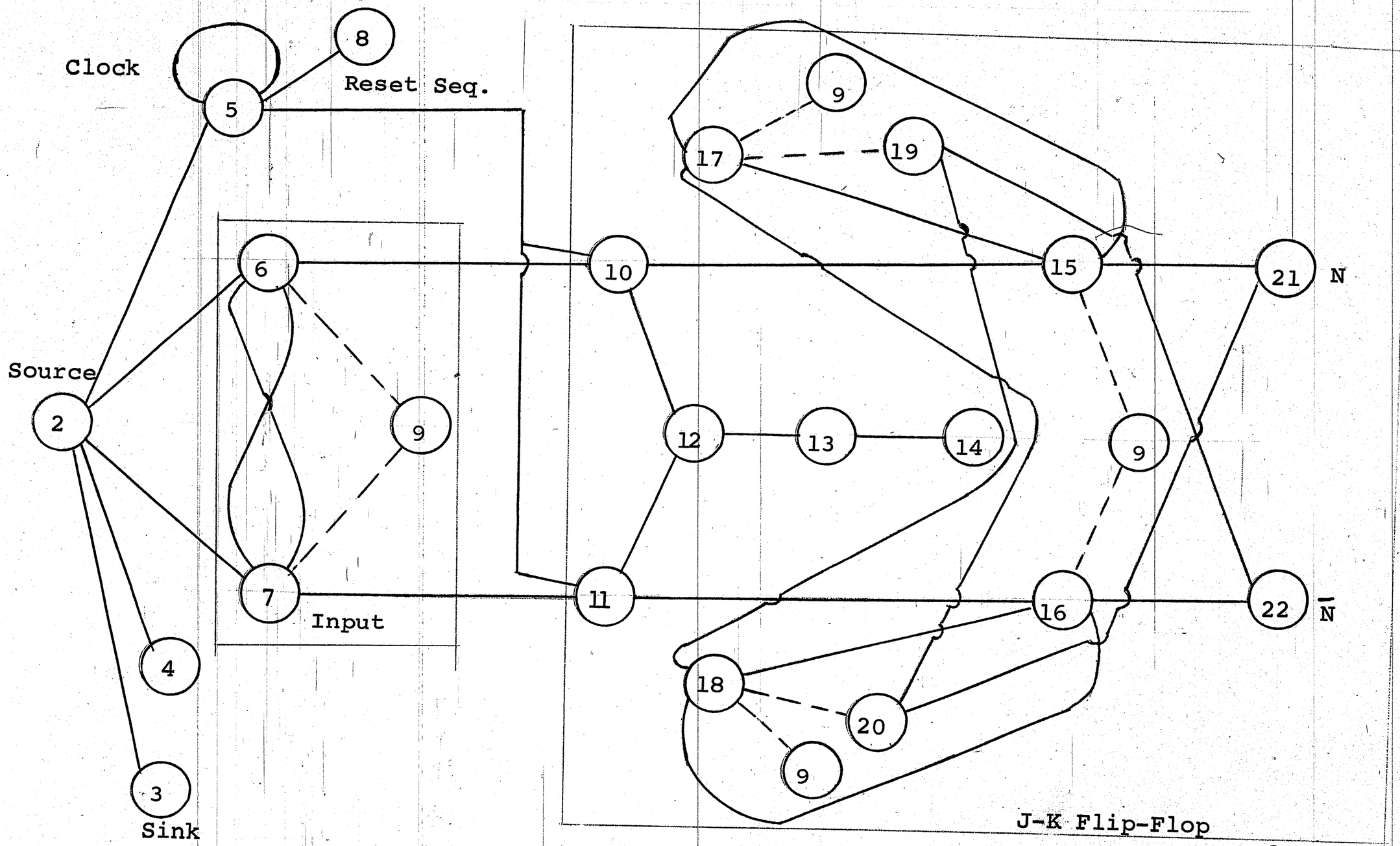
Figure XXXIV is the GERT model of a j-k flip-flop modeled with all combinations of inputs imposed. The emperical output conforms precisely to the truth table function required of such a model. See Appendix C.

A Large-Scale Circuit

In an article (5), A. J. Dachel developed a clocked sequential circuit with two s-r flip-flops, 14 gates and numerous feedback lines. Figure XXXV is a drawing of this circuit. The circuit has three possible states, one input line, and one output node.

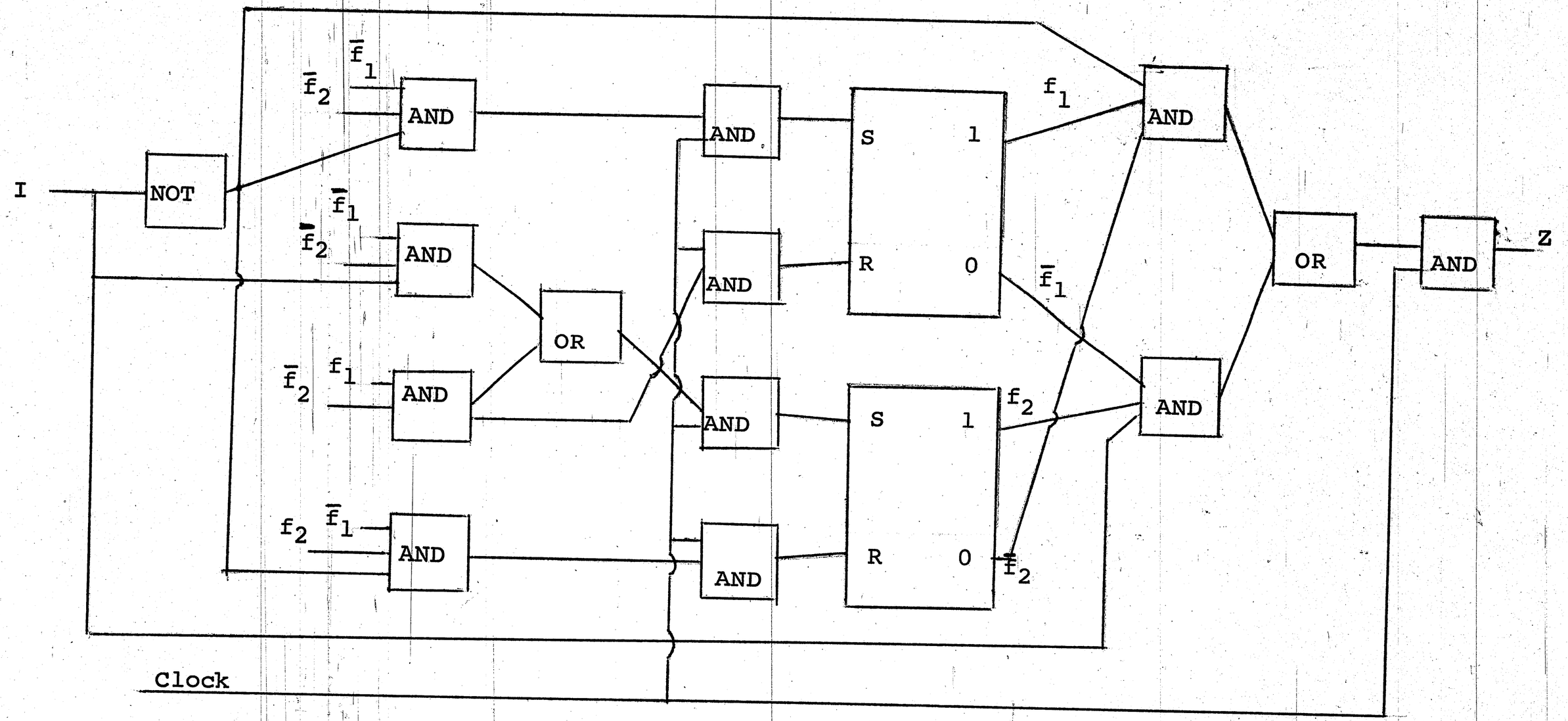
The GERT representation of this circuit is shown in Figure XXXVI. In this figure, the initial state of each flip-flop is state 0.

Mr. Dachel listed in his paper the proper operation of the circuit. The GERT output matches his results. See Appendix D.



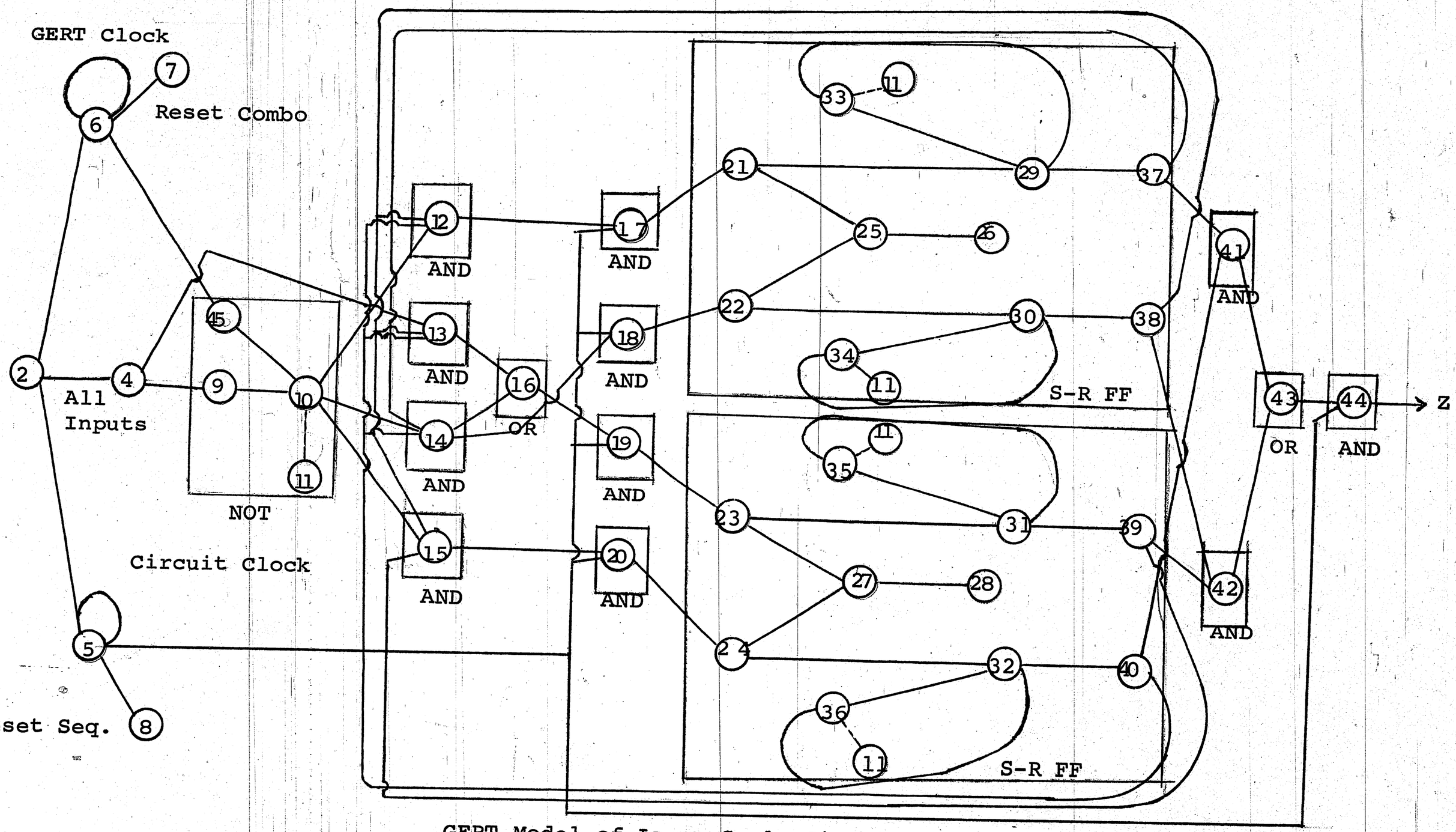
GERT Model of J-K Flip-Flop As Tested

Figure XXXIV.



Large-Scale Circuit

Figure XXXV.



GERT Model of Large-Scale Circuit

Figure XXXVI.

FAULT DETECTION

The essence of fault detection procedures is to test the operation of a properly operating circuit, then impose faults at various gates, and compare the results. That is, for each input combination, the corresponding output for a properly operating circuit is compared to the output of the circuit with a fault. If the two outputs are the same, the fault is undetectable. If a difference exists, the fault is detectable (but not necessarily uniquely detectable). In this way, a complete set of fault information can be generated.

The proper operation of a logic circuit can be demonstrated by GERT simulation. However, to generate fault information, the network must be simulated with faults interferring with its operation. The consequence of each fault can then be determined.

The testing of combinational circuits is somewhat different than that of sequential circuits. However, the procedure outlined above is basically applicable to both types; that is, simulate without faults, then with, and observe if a difference exists. Nevertheless, due to the dissimilarities in logic circuits, the two major types will be considered for

fault detection separately.

Combinational Circuits

Since the output of a combinational circuit is solely dependent upon the inputs to that circuit, the result of each input-fault combination can be considered separately. That is, each test as just described would be a simulation of the circuit. The number of simulations (or tests) that need be run is obviously dependent upon the amount of information desired. The quantity of information developed by the program can range from a complete set to that information caused by a few input-fault combinations. The faults under consideration in most algorithms are of three types: stuck-at-1, stuck-at-0, and line faults. These faults can be programmed in the following manner:

(a) "Stuck-at-1". This occurs when a gate produces an output of 1 under all combinations of input. To accomplish this effect, the GERT node corresponding to the output side of the "stuck-at-1" gate is defined as an input node for the GERT network. This will guarantee a "1" output from the

gate independent of the input.

(b) "Stuck-at-0". This occurs when a gate produces an output of 0 independent of the input to the gate. This can be accomplished by making sure that the GERT node corresponding to the output side of the gate is never realized. Therefore, if the number of releases for this output node is set sufficiently high so that it will never be realized, the "stuck-at-0" condition can be simulated.

(c) Line Faults. Line faults such as "missing wires" can be simulated by modifying the GERT activity corresponding to the line in question. For example, the "missing wire" fault can be simulated by removing the GERT activity associated with the wire in the network. The assumption is made that only one fault can occur at a time. However, GERT may prove useful in extending that assumption.

To generate a complete fault table, the program must simulate all input combinations, first without faults, then once for each possible fault. If there are n-inputs and m possible faults, the network must be simulated $(m+1)2^n$.

Obviously, this is the most inefficient and most cumbersome technique. There is much room for improvement in both the algorithm and the programming technique.

The fault detection procedure described above is too restricting to be of practical value to large microcircuits. However, there are more efficient methods of generating fault information. An algorithm developed by Armstrong (3) eliminates an extensive amount of computer activity in generating the set of detectable faults. His method isolates the effect of a fault by a technique called "path sensitizing". This method significantly reduces the total number of simulations required to generate a complete set of detectable faults. If the circuit in question had n-inputs and m possible faults, this technique would require $2m+2^n$ simulations, whereas, before $(m+1)2^n$ simulations were needed. However, "path sensitizing" does present some programming difficulties not encountered in the algorithm previously discussed. Nevertheless, the net effect would be to dramatically lessen the work that must be performed by the program, and, in that way, expand its usefulness to larger circuits.

No matter what algorithm is employed, once the network has been simulated, the problem reduces to a data-processing problem of discovering and discarding redundancies.

Sequential Circuits

The output of a sequential circuit is not only dependent upon the inputs but also upon the state of the flip-flops. Thus, the sequence of inputs is critical to the analysis. The checking sequence of a sequential circuit consists of a segment to drive the circuit to an initial state and a segment to determine the response of each state to the distinguishing sequence. A distinguishing sequence is one wherein a unique response is obtained for each state.

Therefore, in testing a sequential circuit, a sequence of inputs are imposed so as to drive the circuit into an initial state, then a distinguishing sequence is applied. The testing is completed when a Boolean difference in the outputs (as compared to the output of a properly operating circuit) is detected.

To develop fault information with GERT, the properly operating circuit can be quickly simulated to generate the correct outputs. The program can then impose a fault and impose the input sequence specified. The program will continuously check the output of the faulty circuit against the proper output. Once a difference is detected, the testing is completed for that fault. Then, a new fault is imposed and the sequence is again begun. This continues until all faults (or all faults requested) have been tested for. Once this phase is completed, the problem again reduces to a data-processing problem.

The programming of faults must be somewhat different for sequential circuits due to the memory and feedback arrangements. The types of faults are the same with the addition of faults concerning flip-flops. The faults can be programmed in the following manner:

(a) "Stuck-at-1". To accomplish this, the faulty node will transmit a realization from the GERT clock at each cycle, thus assuring a 1 output for each set of inputs.

(b) "Stuck-at-0". This can be executed by setting the number

of releases for the gates output node extremely high so that it will never be realized.

(c) Line Faults. The "missing wire" fault can be simulated by simply removing the GERT activity associated with that wire.

(d) "Flip-flop will not set". A flip-flop will not set when no matter what inputs it receives it remains in state 0. Thus, the "1" output is "stuck-at-0" and the "0" output is "stuck-at-1". Each line can be programmed precisely in that fashion.

(e) "Flip-flop will not reset". A flip-flop will not reset when no matter what inputs it receives it remains in state 1. Thus, the "1" output is "stuck-at-1" and the "0" output is "stuck-at-0". Each line can be programmed precisely in that fashion.

The assumptions are made that only one fault exists at a time and that asynchronous operation is not caused by the fault. Neither assumption is restricting to GERT.

The simulation of sequential circuits for fault detection can now be programmed. The GERTS II Simulator, since it is a

Fortran based program, is easily modified. The task is (1) to adapt the input mechanism to assimulate data that will describe an electronic circuit, (2) redefine the simulation procedure to inspect the operation of the circuit with and without faults in accordance with a prescribed algorithm, and (3) to evaluate and reveal the results of the investigation.

The Wide Clock

As mentioned earlier, a wide clock could cause a flip-flop to complete its transition from present state to next state before the clock pulse has ended. This would result in asynchronous operation. A single input could result in two transitions and two pulses at the output node.

This peculiarity poses no insurmountable problems for GERT. The wide clock could be represented by two clock pulses. The inputs associated with the clock pulse must also be duplicated. The essence of this procedure is to register one set of inputs at the input node for two consecutive clock pulses. The recording and reporting technique would have to be adapted for the wide clock, but this would not cause extreme difficulties.

It would be safe to predict that other causes of race conditions, such as fast or slow gates, can be modeled in GERT to generate fault information.

SUMMARY AND CONCLUSIONS

The primary intent of this research was exploratory in nature. That is, the desire was to investigate the applicability of GERT to the field of fault analysis of microcircuits. The fulfillment of this objective became quickly apparent and the generation of specific results consequently precipitated with each additional increment of inquiry.

This thesis demonstrated that electronic logic circuits can be modeled using GERT. Their proper and faulty operation can be simulated using a modified GERTS II Simulator. This was accomplished with the sequential completion of three thresholds.

First, the concept of using GERT as a model for the simulation of microcircuits was formulated. The simulation interface was bridged by the following analogy. If a GERT activity is realized, the corresponding electronic line is at the value "1". If the GERT activity is not realized, the line has the value "0". With this criterion GERT could be used to simulate electronic circuits.

Second, the individual gates were modeled. That is, GERT models were constructed such that the Boolean function indigenous to each type of gate was properly represented with regards to the analogy detailed in the last paragraph. Of course, at this stage, the analysis had to be sub-divided according to the major classification of circuits, combinational and sequential. However, the procedures were basically the same for each.

Lastly, the GERT models were integrated in such a way as to truely represent the operation of the circuit under consideration. The objective was amenably aided by the previous accomplishments. That is, the GERT models developed for each individual gate were quite congenial to the completion of this step.

Thus, this thesis has shown that GERT has the capability of being concommitantly valuable to fault detection algorithms. The development of a universal computer program to simulate logic circuits and test their fault characteristics would clearly extend the knowledge and the ability to generate new

knowledge in the area of fault detection. This thesis has shown that GERT has the facility to designate that goal attainable.

AREAS FOR FURTHER STUDY

The use of GERT as a circuit simulator is limited in that the present GERT program was not designed with this application in mind. Since the purpose of utilizing GERT in this fashion is to provide a quick and easy programming technique for fault analysis, the improvement of the GERT program with this one particular application in mind would be obviously beneficial.

Presently, the major obsticle to realizing GERTS true effectiveness is the number of activities that must be scheduled. If the number of activities could be significantly reduced the utility of the GERT Simulator would be tremendously enhanced. There are several ways to accomplish this. I have proposed a few:

1. Design simpler (having fewer activities) models of individual gates.

2. Eliminate the need for the GERT clock by having an inherent sequencing procedure.

3. Develop GERT "black box" representations of gates. For example, develop a GERT node that responds precisely

as a NAND or NOR, etc.

Other improvements, as mentioned in the text of the thesis, are attainable by programming more efficient algorithms such as Armstrong's technique of "path sensitizing". This poses several programming problems, however, the net effect would be beneficial.

With these improvements, it would clearly establish GERT as a pragmatic partner in fault detection research.

This thesis has been concerned solely with fault detection. However, it is not unreasonable to assume that a GERT program developed to model logic circuits could be useful to other areas of electronic research now under study or otherwise. The search for new applications may prove quite fruitful.

APPENDIX A

Modulo "8" Counter

The counter considered has three toggle flip-flops. The circuit is shown in Figure XXX and the GERT representation is in Figure XXXI. The counter operates in the manner described in Figure XXXVII. The three outputs, y_1, y_2, y_3, are represented by GERT nodes 22, 23, 24 respectively. GERT node 2 is the clock pulse.

The computer output confirms the validity of the model. Its output is the values of the output at the time of each clock pulse.

y_3	y_2	y_1
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1
0	0	0

Counter Operation - Response to Clock Pulses

Figure XXXVII

The GERTS II Simulator used was essentially the same as listed in reference (14). The GASP subroutine has been modified as shown. All runs used the same basic listing. Therefore, a copy of the Modulo "8" Counter was included in this thesis; copies of the other runs were not included.

```
      SUBROUTINE GASP(NSET,QSET)
      DIMENSION NSET(1),QSET(1)
      COMMON ID,IM,MFA,MSTOP,MX,MXC,NCLCT,NHIST,NPRNT,NCRDR,
     1NOQ,NPRMS,NRUN,NRUNS,ISEED,TNOW,MXX,IMM,NYR,SEED,
     2MAXQS,MAXNS,ATRIB(1),JTRIB(5),NAME(6),NPROJ,MON,NDAY,J
      COMMON MFE(999),MLE(999),NQ(999),PARAM(300,4),SUMA(100
      COMMON NSINK(20),NSKST,NSRC,XLOW(20),NRELP(999),NREL(9
     1NSKSR,NSORC(20),NTYPE(999),WIDTH(20),TOTIM,NCTS,KOUNT(
      COMMON NPO(999),NFTBU(999),NABA(999)
      DIMENSION NCLER(100),LNUM(100),LVAL(100)
      DIMENSION LLL(100)
      DO 618 I=1,100
  618 LVAL(I)=0
      DO 620 I=2,500
  620 NRELP(I)=0
      READ(NCRDR,619)(LNUM(I),I=1,8)
  619 FORMAT(10I8)
      CALL DATAN(NSET,QSET)
      KJZ=3
      NREL(4)=NREL(4)-1
      NREL(12)=NREL(12)-1
      NREL(18)=NREL(18)-1
      NCLER(1)=10
      NCLER(2)=16
      NBT=2
C
C*****OBTAIN NEXT EVENT WHICH IS FIRST ENTRY IN FILE 1.  ATR
C*****EVENT TIME, JTRIB(1) IS EVENT CODE
C
   11 CALL RMOVE(MFE(1),1,NSET,QSET)
      TNOW=ATRIB(1)
      NEND=JTRIB(1)
      IF(NEND-8)603,604,603
  604 DO 602 I=1,NBT
      K=NCLER(I)
  602 NREL(K)=NRELP(K)
      WRITE(NPRNT,609)
  609 FORMAT(////)
      DO 606 J=1,3
      I=LNUM(J)
  606 LLL(J)=LVAL(I)
      WRITE(NPRNT,605)(LLL(J),J=1,KJZ)
  605 FORMAT(10X,10I10)
      DO 607 I=1,100
  607 LVAL(I)=0
  603 CONTINUE
C
C*****IF END NODE IS ZERO START NETWORK BY SCHEDULING ACTIV
C*****FROM EACH SOURCE NODE
C
      IF(NEND)4,4,5
    4 DO 10 N=1,NSRC
      M=NSORC(N)
      CALL SCHAT(M,NSET,QSET)
   10 CONTINUE
      GO TO 42
```

```
C*****REDUCE NO. OF RELEASES FOR END NODE BY 1.
C*****TEST TO SEE IF NODE IS RELEASED.                          80.
C
     5 NREL(NEND)=NREL(NEND)-1
       IF(NPROJ) 511,512,512
   511 IF(NRUN-NABA(IMN)) 512,514,514
   514 IMN1=IMN-1
       IF(NRUN-NABA(IMN1)) 515,515,512
   515 WRITE(NPRNT,598) TNOW,JTRIB
   598 FORMAT (5X,7HAT TIME,F8.2,17H ACTIVITY ON NODE,I5,16H
      1TES,415,13H WAS REALIZED)
   512 IF(JTRIB(4)) 665,665,60
    60 L=JTRIB(4)
       KOUNT(L)=KOUNT(L)+1
   665 IF(JTRIB(5)) 65,65,66
    66 JJJ=JTRIB(5)
       IF(NPO(JJJ)) 65,65,67
    67 NPOS=NPO(JJJ)
    69 NDN=NABA(NPOS)
       NFTBU(NDN)=NABA(NPOS+1)
       IF(NABA(NPOS+2)) 65,65,68
    68 NPOS=NPOS +2
       GO TO 69
    65 IF(NREL(NEND))9,7,42
C
C*****TEST TO SEE IF NODE CAN BE RELEASED MORE THAN ONCE.
C
     9 IF(NTYPE(NEND)-2)42,42,50
C
C*****TEST TO SEE IF STATISTICS ARE TO BE COLLECTED ON THE
C
     7 LVAL(NEND)=1
       DO 8 K=1,NSKS
       IF(NEND-NSINK(K))8,39,8
    39 IF(JSINK(K))50,50,40
     8 CONTINUE
    50 NT=NTYPE(NEND)
       GO TO (52,52,52,52,52,52,54,54),NT
    54 KCOL=NFIND(NEND,1,1,NSET,QSET)
        IF(KCOL) 52,52,56
    56 CALL RMOVE(KCOL,1,NSET,QSET)
       GO TO 54
    52 CALL SCHAT(NEND,NSET,QSET)
       GO TO (42,42,42,42,51,51,51,51),NT
    51 NREL(NEND)=NRELP(NEND)
       GO TO 42
C
C*****REDUCE NO. OF SINK NODES REALIZED BY ONE.  COLLECT S
C
    40 IF(K-NPD) 440,440,441
   440 NSKSR=NSKSR-1
   441 JSINK(K)=0
       CALL HISTO(TNOW,XLOW(K),WIDTH(K),K)
       NL=K+(K-1)*NCTS
       CALL COLCT(TNOW,NL,NSET,QSET)
       IF(NCTS) 45,45,46
    46 DO 70 I=1,NCTS
       NL=NL+1
```

```
      XC=KOUNT(I)
   70 CALL COLCT(XC,NL,NSET,QSET)
C
C*****TEST TO SEE IF NETWORK SIMULATION IS COMPLETE.
C
   45 IF(NSKSR) 22,100,999
  999 IF(NQ(NEND))22,42,50
   22 CALL ERROR(22,NSET,QSET)
C
C*****NETWORK HAS BEEN SIMULATED ONE MORE TIME.  ADD TNOW T
C
  100 TOTIM=TOTIM+TNOW
C
C*****TEST TO SEE IF ALL RUNS HAVE BEEN MADE.
C
      IF(NRUNS-NRUN)110,110,115
C
C*****REINITIALE FOR ANOTHER RUN BY REMOVING ALL EVENTS FRC
C*****FILE AND RESETTING NETWORK VALUES.
C
  115 IF(NQ(1))22,120,116
  116 CALL RMOVE(MFE(1),1,NSET,QSET)
      GO TO 115
  120 NRUN=NRUN+1
      JTRIB(1)=0
      ATRIB(1)=0.
      CALL FILEM(1,NSET,QSET)
      TNOW=0.
      NSKSR=NSKST
      DO 80 K=1,NSKS
   80 JSINK(K)=1
      DO 85 J=1,NCTS
   85 KOUNT(J)=0
      DO 37 IJ=1,NN
      NFTBU(IJ)=IJ
   37 NREL(IJ)=NRELP(IJ)
      GO TO 42
  110 MSTOP=-1
      NRUNS=1
      TNOW=TOTIM
C
C*****IS IT TIME TO STOP
C
   42 IF(MSTOP)44,11,11
   44 CALL SUMRY(NSET,QSET)
      RETURN
      END
```

ACTIVITY PARAMETERS

PARAMETER NUMBER	PARAMETERS 1	2	3
1	0.0000	-0.0000	-0.0000
2	4.5000	-0.0000	-0.0000
3	5.0000	-0.0000	-0.0000
4	10.0000	-0.0000	-0.0000
5	8.0000	-0.0000	-0.0000
6	103.0000	-0.0000	-0.0000
7	2.0000	-0.0000	-0.0000

NETWORK DESCRIPTION

START NODE	END NODE	PARAMETER NUMBER	DISTRIBUTION TYPE	COUNT TYPE	ACTIVITY NUMBER	PROBABILITY
2	2	4	1	-0	-0	1.0000
2	8	2	1	0	4	1.0000
2	3	1	1	-0	-0	1.0000
2	10	1	1	-0	-0	1.0000
2	16	1	1	-0	-0	1.0000
3	4	1	1	0	1	1.0000
4	9	3	1	-0	-0	1.0000
5	6	1	1	-0	-0	1.0000
6	9	7	1	-0	-0	1.0000
9	5	5	1	-0	-0	1.0000
9	10	1	1	-0	-0	1.0000
9	16	1	1	-0	-0	1.0000
9	22	1	1	-0	-0	1.0000
10	11	1	1	-0	-0	1.0000
11	12	1	1	-0	-0	1.0000
12	15	3	1	0	2	1.0000
13	14	1	1	-0	-0	1.0000
14	15	7	1	-0	-0	1.0000
15	13	5	1	-0	-0	1.0000
15	16	1	1	-0	-0	1.0000
15	23	1	1	-0	-0	1.0000
16	17	1	1	-0	-0	1.0000
17	18	1	1	-0	-0	1.0000
18	19	3	1	0	3	1.0000
19	20	5	1	-0	-0	1.0000
19	24	1	1	-0	-0	1.0000
20	21	1	1	-0	-0	1.0000
21	19	7	1	-0	-0	1.0000
26	2	1	1	-0	-0	1.0000
26	25	6	1	-0	-0	1.0000

NETWORK MODIFICATIONS

ACTIVITY	NODE	FILE	NODE	FILE	NODE	FILE	NODE	FILE	NODE	FILE
1	6	7								
2	14	7								
3	21	7								
4	6	6	14	14	21	21				

HIGHEST NODE NUMBER IS 26
NUMBER OF SOURCE NODES IS 1
NUMBER OF SINK NODES IS 1
NUMBER OF NODES TO REALIZE THE NETWORK IS 1
STATISTICS COLLECTED ON 1 NODES
NUMBER OF PARAMETER SETS IS 7
INITIAL RANDOM NUMBER IS -0 -0.0000

SOURCE NODE NUMBERS
26

SINK NODE NUMBERS
25

NODE	NUMBER RELEASES	NODE TYPE
2	1	5
3	1	5
4	2	5
5	1	5
6	1	5
7	1	5
8	1	5
9	1	5
10	2	5
11	1	5
12	2	5
13	1	5
14	1	5
15	1	5
16	3	5
17	1	5
18	2	5
19	1	5
20	1	5
21	1	5
22	1	5
23	1	5
24	1	5
25	1	5
26	1	5

Y_3	Y_2	Y_1
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1
0	0	0
0	0	1

APPENDIX B

Shift Register

The two flip-flop shift register under consideration is shown in Figure XXXII. The GERT representation is in Figure XXXIII. The flip-flops have been initialized to state 0. The proper operation is shown in Figure XXXVIII. The inputs to the system were alternatively 1 then 0.

The node designated to reset the combinational elements is node 36. The time associated with the realization of this node is 9.99. It should be 4.5 to guarantee proper operation of a circuit. However, due to the nature of this circuit, it was not critical.

The computer output yields results conforming to the desired output listed in Figure XXXVIII. Nodes 6, 34, 35, 32, 33 were monitored for the values of x, ff1, and ff2.

	ff_1		ff_2	
x	1	0	1	0
1	0	1	0	1
0	1	0	0	1
1	0	1	1	0
0	1	0	0	1

Operation of Shift Register

Figure XXXVIII

GERT SIMULATION PROJECT 2 BY RICCIO
DATE 4/ 3/ 1970

NETWORK DESCRIPTION

START NODE	END NODE	PARAMETER NUMBER	DISTRIBUTION TYPE	COUNT TYPE	ACTIVITY NUMBER
2	37	14	1	-0	-0
2	33	1	1	-0	-0
2	35	1	1	-0	-0
2	3	12	1	-0	-0
2	4	1	1	-0	-0
2	5	1	1	-0	-0
2	6	1	1	-0	-0
4	36	15	1	0	5
4	4	10	1	-0	-0
4	7	2	1	-0	-0
4	14	3	1	-0	-0
4	15	3	1	-0	-0
4	24	4	1	-0	-0
4	25	4	1	-0	-0
5	8	6	1	0	4
5	5	10	1	-0	-0
5	12	1	1	-0	-0
5	13	1	1	-0	-0
5	22	1	1	-0	-0
5	23	1	1	-0	-0
6	6	11	1	-0	-0
6	9	1	1	-0	-0
6	12	1	1	-0	-0
7	10	1	1	-0	-0
9	10	1	1	0	1
10	13	1	1	-0	-0
12	14	1	1	-0	-0

13	15	1	1	-0	-0
14	16	1	1	-0	-0
14	18	14	1	-0	-0
15	16	1	1	-0	-0
15	19	14	1	-0	-0
16	17	1	1	-0	-0
18	20	10	1	0	2
18	34	5	1	-0	-0
19	21	10	1	-0	-0
19	35	5	1	-0	-0
20	18	1	1	-0	-0
21	19	1	1	-0	-0
22	24	1	1	-0	-0
23	25	1	1	-0	-0
24	26	1	1	-0	-0
24	28	14	1	-0	-0
25	26	1	1	-0	-0
25	29	14	1	-0	-0
26	27	1	1	-0	-0
28	30	10	1	0	3
28	32	5	1	-0	-0
29	31	10	1	-0	-0
29	33	5	1	-0	-0
30	28	1	1	-0	-0
31	29	1	1	-0	-0
34	22	1	1	-0	-0
35	23	1	1	-0	-0
37	21	3	1	-0	-0
37	31	4	1	-0	-0

ACTIVITY PARAMETERS

PARAMETER NUMBER	PARAMETERS 1	2	3	4
1	0.0000	-0.0000	-0.0000	-0.0000
2	.0100	-0.0000	-0.0000	-0.0000
3	.0200	-0.0000	-0.0000	-0.0000
4	.0300	-0.0000	-0.0000	-0.0000
5	5.0000	-0.0000	-0.0000	-0.0000
6	4.5000	-0.0000	-0.0000	-0.0000
7	5.0000	-0.0000	-0.0000	-0.0000
8	.4000	-0.0000	-0.0000	-0.0000
9	4.9799	-0.0000	-0.0000	-0.0000
10	10.0000	-0.0000	-0.0000	-0.0000
11	20.0000	-0.0000	-0.0000	-0.0000
12	100.0000	-0.0000	-0.0000	-0.0000
13	4.5999	-0.0000	-0.0000	-0.0000
14	.0000	-0.0000	-0.0000	-0.0000
15	9.9900	-0.0000	-0.0000	-0.0000
16	.0201	-0.0000	-0.0000	-0.0000
17	.4001	-0.0000	-0.0000	-0.0000

NETWORK MODIFICATIONS

ACTIVITY	NODE	FILE	NODE	FILE	NODE	FILE	NODE	FILE	NODE	FILE	NODE	FILE	NODE	FILE
1	10	11												
2	20	11	21	11										
3	30	11	31	11										
4	20	20	21	21	30	30	31	31						
5	10	10												

HIGHEST NODE NUMBER IS 37
NUMBER OF SOURCE NODES IS 1
NUMBER OF SINK NODES IS 1
NUMBER OF NODES TO REALIZE THE NETWORK IS 1
STATISTICS COLLECTED ON 1 NODES
NUMBER OF PARAMETER SETS IS 17
INITIAL RANDOM NUMBER IS -0 -0.0000

SOURCE NODE NUMBERS
2

SINK NODE NUMBERS
3

NODE	NUMBER RELEASES	NODE TYPE
2	1	5
3	1	5
4	1	5
5	1	5
6	1	5
7	1	5
8	1	5
9	1	5
10	1	5
11	1	5
12	2	5
13	2	5
14	2	5
15	2	5
16	1	5
17	2	5
18	1	5
19	1	5
20	1	5
21	1	5
22	2	5
23	2	5
24	2	5
25	2	5
26	1	5
27	1	5
28	1	5
29	1	5
30	1	5
31	1	5
32	1	5
33	1	5
34	1	5
35	1	5
36	1	5
37	1	5

X	FF_1		FF_2	
	1	0	1	0
1	0	1	0	1
0	1	0	0	1
1	0	1	1	0
0	1	0	0	1
1	0	1	1	0
0	1	0	0	1
1	0	1	1	0
0	1	0	0	1

APPENDIX C

Test of j-k Flip-Flop

A test of the j-k flip-flop model was developed. All combinations of inputs were imposed on the inputs to the model and the outputs were recorded. Figure XXXIV is the GERT model of a j-k flip-flop with inputs generated externally. Nodes 6 and 7 are the j and k inputs respectively while nodes 21 and 22 are the outputs.

The correct output is listed in Figure XXXIX. The GERT results are identical with the exception of the circled data. Due to the peculiarity of the GERT modification procedure, although node 9 had replaced node 7, node 7 is still considered realized. However, that impulse recorded by the program was not allowed to propagate. It died in node 9. Thus, instead of a (0, 1) input, in reality a (0, 0) input was absorbed by the flip-flop. Therefore, the GERT results are correct. This is pointed out in the GERT output by having the (0, 1) input circled.

Inputs		Present State	
1	0	0	1
0	1	1	0
1	1	0	1
1	0	1	0
0	0	1	0
0	0	1	0

j-k Flip-Flop Operation

Figure XXXIX

START NODE	END NODE	PARAMETER NUMBER	DISTRIBUTION TYPE	COUNT TYPE	ACTIVITY NUMBER	PROBABILITY
2	3	12	1	0	-0	1.0000
2	4	11	1	0	1	1.0000
2	5	1	1	-0	-0	1.0000
2	6	1	1	-0	-0	1.0000
2	7	9	1	-0	-0	1.0000
2	23	2	1	-0	-0	1.0000
2	22	1	1	-0	-0	1.0000
5	5	9	1	-0	-0	1.0000
5	8	7	1	0	4	1.0000
5	10	1	1	-0	-0	1.0000
5	11	1	1	-0	-0	1.0000
6	7	10	1	-0	-0	1.0000
6	10	1	1	-0	-0	1.0000
7	6	9	1	-0	-0	1.0000
7	11	1	1	-0	-0	1.0000
10	12	1	1	-0	-0	1.0000
10	15	2	1	-0	-0	1.0000
11	12	1	1	-0	-0	1.0000
11	16	2	1	-0	-0	1.0000
12	13	1	1	-0	-0	1.0000
13	14	1	1	0	2	1.0000
15	17	9	1	0	3	1.0000
15	21	5	1	-0	-0	1.0000
16	18	9	1	-0	-0	1.0000
16	22	5	1	-0	-0	1.0000
17	15	1	1	-0	-0	1.0000
18	16	1	1	-0	-0	1.0000
19	22	5	1	-0	-0	1.0000
19	18	9	1	-0	-0	1.0000
20	21	5	1	-0	-0	1.0000
20	17	9	1	-0	-0	1.0000
23	18	1	1	-0	-0	1.0000

PARAMETER NUMBER	PARAMETERS 1	2	3	4
1	0.0000	-0.0000	-0.0000	-0.0000
2	.0010	-0.0000	-0.0000	-0.0000
3	.0100	-0.0000	-0.0000	-0.0000
4	4.9900	-0.0000	-0.0000	-0.0000
5	4.9990	-0.0000	-0.0000	-0.0000
6	4.9890	-0.0000	-0.0000	-0.0000
7	4.5000	-0.0000	-0.0000	-0.0000
8	5.0000	-0.0000	-0.0000	-0.0000
9	10.0000	-0.0000	-0.0000	-0.0000
10	20.0000	-0.0000	-0.0000	-0.0000
11	39.0000	-0.0000	-0.0000	-0.0000
12	59.0000	-0.0000	-0.0000	-0.0000

NETWORK MODIFICATIONS

ACTIVITY	NODE	FILE	NODE	FILE	NODE	FILE	NODE	FILE	NODE	FILE	NODE	FILE	NODE	FILE
1	6	9	7	9										
2	17	9	18	9										
3	17	19	18	20	15	9	16	9						
4	6	6	7	7	17	17	18	18	15	15	16	16		

HIGHEST NODE NUMBER IS 23
NUMBER OF SOURCE NODES IS 1
NUMBER OF SINK NODES IS 1
NUMBER OF NODES TO REALIZE THE NETWORK IS 1
STATISTICS COLLECTED ON 1 NODES
NUMBER OF PARAMETER SETS IS 12
INITIAL RANDOM NUMBER IS -0 -0.0000

SOURCE NODE NUMBERS
2

SINK NODE NUMBERS
3

NODE	NUMBER RELEASES	NODE TYPE
2	1	5
3	1	5
4	1	5
5	1	5
6	1	5
7	1	5
8	1	5
9	1	5
10	2	5
11	2	5
12	1	5
13	2	5
14	1	5
15	1	5
16	1	5
17	1	5
18	1	5
19	1	5
20	1	5
21	1	5
22	1	5
23	-0	5

INPUTS		PRESENT STATE	
1	0	0	1
0	1	1	0
1	1	0	1
1	0	1	0
0	1	1	0
0	0	1	0

APPENDIX D

Large-Scale Circuit

The circuit developed by Dachel in (5) was programmed. This circuit is shown in Figure XXXV and its GERT representation is shown in Figure XXXVI. The initial state of each flip-flop is 0. The proper output is shown in Figure XL.

The results verify the validity of the model. Again, the time associated with the resetting of combinational elements is 9.99, but should be 4.5. However, as before, this was not critical.

Initial State (0, 0)

	Present State				
	ff_1		ff_2		
I	1	0	1	0	z
1	0	1	0	1	0
0	0	1	1	0	0
1	0	1	0	1	0
0	0	1	1	0	0
1	0	1	0	1	0
1	0	1	1	0	1
0	0	1	1	0	0
0	0	1	0	1	0
1	1	0	0	1	0
0	1	0	0	1	1
0	0	1	1	0	0
1	0	1	0	1	0
0	0	1	1	0	0
1	0	1	0	1	0
0	0	1	1	0	0
1	0	1	0	1	0
0	0	1	1	0	0
1	0	1	0	1	0
0	0	1	1	0	0
1	0	1	0	1	0
0	0	1	1	0	0
1	0	1	0	1	0
1	0	1	1	0	1
1	0	1	1	0	1

Operation of Large-Scale Circuit

Figure XL

START NODE	END NODE	PARAMETER NUMBER	DISTRIBUTION TYPE	COUNT TYPE	ACTIVITY NUMBER	PROBABILITY
2	3	25	1	-0	-0	1.0000
2	4	1	1	-0	-0	1.0000
2	5	1	1	-0	-0	1.0000
2	6	1	1	-0	-0	1.0000
2	4	9	1	-0	-0	1.0000
2	4	10	1	-0	-0	1.0000
2	4	11	1	-0	-0	1.0000
2	4	12	1	-0	-0	1.0000
2	4	13	1	-0	-0	1.0000
2	4	14	1	-0	-0	1.0000
2	4	15	1	-0	-0	1.0000
2	4	16	1	-0	-0	1.0000
2	4	17	1	-0	-0	1.0000
2	4	18	1	-0	-0	1.0000
2	4	19	1	-0	-0	1.0000
2	4	20	1	-0	-0	1.0000
2	4	21	1	-0	-0	1.0000
2	4	22	1	-0	-0	1.0000
2	4	23	1	-0	-0	1.0000
2	4	24	1	-0	-0	1.0000
2	38	1	1	-0	-0	1.0000
2	46	1	1	-0	-0	1.0000
2	40	1	1	-0	-0	1.0000
4	9	1	1	-0	-0	1.0000
4	13	1	1	-0	-0	1.0000
4	42	1	1	-0	-0	1.0000
5	5	8	1	-0	-0	1.0000
5	8	5	1	0	4	1.0000
5	17	1	1	-0	-0	1.0000
5	18	1	1	-0	-0	1.0000
5	19	1	1	-0	-0	1.0000
5	20	1	1	-0	-0	1.0000
5	44	1	1	-0	-0	1.0000

6	6	8	1	-0	-0	1.0000
6	45	3	1	-0	-0	1.0000
6	7	7	1	0	5	1.0000
6	21	4	1	-0	-0	1.0000
6	22	4	1	-0	-0	1.0000
6	23	4	1	-0	-0	1.0000
6	24	4	1	-0	-0	1.0000
9	10	1	1	0	1	1.0000
10	41	1	1	-0	-0	1.0000
10	12	1	1	-0	-0	1.0000
10	14	1	1	-0	-0	1.0000
10	15	1	1	-0	-0	1.0000
12	17	1	1	-0	-0	1.0000
13	16	1	1	-0	-0	1.0000
14	16	1	1	-0	-0	1.0000
14	18	1	1	-0	-0	1.0000
15	20	1	1	-0	-0	1.0000
16	19	1	1	-0	-0	1.0000
17	21	1	1	-0	-0	1.0000
18	22	1	1	-0	-0	1.0000
19	23	1	1	-0	-0	1.0000
20	24	1	1	-0	-0	1.0000
21	25	1	1	-0	-0	1.0000
21	29	2	1	-0	-0	1.0000
22	25	1	1	-0	-0	1.0000
22	30	2	1	-0	-0	1.0000
23	27	1	1	-0	-0	1.0000
23	31	2	1	-0	-0	1.0000
24	27	1	1	-0	-0	1.0000
24	32	2	1	-0	-0	1.0000
25	26	1	1	-0	-0	1.0000
27	28	1	1	0	2	1.0000
29	33	8	1	0	3	1.0000
29	37	6	1	-0	-0	1.0000
30	34	8	1	-0	-0	1.0000
30	38	6	1	-0	-0	1.0000

31	35	8	1	-0	-0	1.0000
31	39	6	1	-0	-0	1.0000
32	36	8	1	-0	-0	1.0000
32	40	6	1	-0	-0	1.0000
33	29	1	1	-0	-0	1.0000
34	30	1	1	-0	-0	1.0000
35	31	1	1	-0	-0	1.0000
36	32	1	1	-0	-0	1.0000
37	41	1	1	-0	-0	1.0000
37	14	1	1	-0	-0	1.0000
38	42	1	1	-0	-0	1.0000
38	12	1	1	-0	-0	1.0000
38	13	1	1	-0	-0	1.0000
38	15	1	1	-0	-0	1.0000
39	42	1	1	-0	-0	1.0000
39	15	1	1	-0	-0	1.0000
40	41	1	1	-0	-0	1.0000
40	12	1	1	-0	-0	1.0000
40	13	1	1	-0	-0	1.0000
40	14	1	1	-0	-0	1.0000
41	43	1	1	-0	-0	1.0000
42	43	1	1	-0	-0	1.0000
43	44	1	1	-0	-0	1.0000
45	10	1	1	-0	-0	1.0000
46	47	4	1	-0	-0	1.0000
47	34	2	1	-0	-0	1.0000
47	36	2	1	-0	-0	1.0000

ACTIVITY PARAMETERS

PARAMETER NUMBER	PARAMETERS 1	2	3	4
1	0.0000	-0.0000	-0.0000	-0.0000
2	.0010	-0.0000	-0.0000	-0.0000
3	.1000	-0.0000	-0.0000	-0.0000
4	.2000	-0.0000	-0.0000	-0.0000
5	4.5000	-0.0000	-0.0000	-0.0000
6	4.9990	-0.0000	-0.0000	-0.0000
7	9.9900	-0.0000	-0.0000	-0.0000
8	10.0000	-0.0000	-0.0000	-0.0000
9	20.0000	-0.0000	-0.0000	-0.0000
10	40.0000	-0.0000	-0.0000	-0.0000
11	50.0000	-0.0000	-0.0000	-0.0000
12	80.0000	-0.0000	-0.0000	-0.0000
13	110.0000	-0.0000	-0.0000	-0.0000
14	130.0000	-0.0000	-0.0000	-0.0000
15	150.0000	-0.0000	-0.0000	-0.0000
16	170.0000	-0.0000	-0.0000	-0.0000
17	180.0000	-0.0000	-0.0000	-0.0000
18	190.0000	-0.0000	-0.0000	-0.0000
19	230.0000	-0.0000	-0.0000	-0.0000
20	260.0000	-0.0000	-0.0000	-0.0000
21	270.0000	-0.0000	-0.0000	-0.0000
22	310.0000	-0.0000	-0.0000	-0.0000
23	340.0000	-0.0000	-0.0000	-0.0000
24	350.0000	-0.0000	-0.0000	-0.0000
25	375.0000	-0.0000	-0.0000	-0.0000

```
HIGHEST NODE NUMBER IS   47
NUMBER OF SOURCE NODES IS     1
NUMBER OF SINK NODES IS     1
NUMBER OF NODES TO REALIZE THE NETWORK IS     1
STATISTICS COLLECTED ON     1 NODES
NUMBER OF PARAMETER SETS IS    25
INITIAL RANDOM NUMBER IS    -0    -0.0000

SOURCE NODE NUMBERS
        2

SINK NODE NUMBERS
        3
```

ACTIVITY	NODE	FILE	NODE	FILE	NODE	FILE	NODE	FILE
1	10	11						
2	33	11	34	11				
3	35	11	36	11				
4	33	33	34	34	35	35	36	36
5	10	10						

NODE	NUMBER RELEASES	NODE TYPE
2	1	5
3	1	5
4	1	5
5	1	5
6	1	5
7	1	5
8	1	5
9	1	5
10	1	5
11	1	5
12	3	5
13	3	5
14	3	5
15	3	5
16	1	5
17	2	5
18	2	5
19	2	5
20	2	5
21	2	5
22	2	5
23	2	5
24	2	5
25	1	5
26	1	5
27	1	5
28	1	5
29	1	5
30	1	5
31	1	5
32	1	5
33	1	5
34	1	5
35	1	5
36	1	5
37	1	5
38	1	5
39	1	5
40	1	5
41	3	5
42	3	5
43	1	5
44	2	5
45	1	5
46	1	5
47	1	5

INITIAL STATE (0,0)

I	PRESENT STATE FF1		PRESENT STATE FF2		
	1	0	1	0	Z
1	0	1	0	1	0
0	0	1	1	0	0
1	0	1	0	1	0
0	0	1	1	0	0
1	0	1	0	1	0
1	0	1	1	0	1
0	0	1	1	0	0
0	0	1	0	1	0
1	1	0	0	1	0

0	1	0	0	1	1
0	0	1	1	0	0
1	0	1	0	1	0
0	0	1	1	0	0
1	0	1	0	1	0
0	0	1	1	0	0
1	0	1	0	1	0
0	0	1	1	0	0
1	0	1	0	1	0
0	0	1	1	0	0
1	0	1	0	1	0

1	0	1	1	0	1
1	0	1	1	0	1
0	0	1	1	0	0
0	0	1	0	1	0
0	1	0	0	1	1
1	0	1	1	0	1
0	0	1	1	0	0

BIBLIOGRAPHY

1. _____, "A Basic Logic Review", Tekscope, December 1969.

2. Amar, V. and A. Condulmari, "Diagnosis of Large Combinational Networks", IEEE Transactions on Electronic Computers, Vol. EC 16, October 1967.

3. Armstrong, D. B., "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic Networks", IEEE Transactions on Electronic Computers, February 1966.

4. Chang, H. Y., "An Algorithm for Selecting an Optimum Set of Diagnostic Tests", IEEE Transactions on Electronic Computers, Vol. EC 14, October 1965.

5. Dachel, A. J., "Fault Analysis of Sequential Circuits", Western Electric Engineering Research Center, Princeton, New Jersey.

6. Dachel, A. J. and P. Nadkarni, "The Generation of Input Vectors for Testing of Combinational Logic Using Roth's Algorithm", Western Electric Research Center, Princeton, New Jersey.

7. Gill, Arthur, Linear Sequential Circuits, McGraw-Hill, New York, 1966.

8. Higonnet, Rene A. and Rene A. Grea, Logical Design of Electrical Circuits, McGraw-Hill, New York, 1958.

9. Hill, Fredrick J. and Gerald R. Peterson, Switching Theory and Logical Design, John Wiley & Sons, 1968.

10. Hornbuckle, Gary D. and Richard N. Spann, "Diagnosis of Single-Gate Failures in Combinational Circuits", IEEE Transactions on Computers, January 1969.

11. Kautz, William H., "Fault Testing and Diagnosis in Combinational Digital Circuits", IEEE Transactions on Computers, April 1968.

12. Keister, William, et. al., The Design of Switching Circuits, D. Van Nostrand, Princeton, New Jersey, 1951.

13. Pritsker, A. A. B. and P. C. Ishmael, User Manual for GERT Exclusive-Or Program, NASA/ERC, NGR 03-111-034, July 1968.

14. Pritsker, A. A. B. and P. C. Ishmael, GERT Simulation Program II (GERTS II), NASA-12-2035, June 1969.

15. Pritsker, A. A. B. and W. W. Happ, "GERT: Graphic Evaluation and Review Technique, Part I Fundamentals", The Journal of Industrial Engineering, Vol. XVII, No. 5, May 1966.

16. Pritsker, A. A. B. and G. E. Whitehouse, "GERT: Graphic Evaluation and Review Technique, Part II Probabilistic and Industrial Engineering Application", The Journal of Industrial Engineering, Vol. XVII, No. 6, June 1966.

17. Roth, J. P., "Diagnosis of Automata Failures a Calculus and a Method", IBM Journal, July 1966.

18. Roth, J. B. et. al., "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits", IEEE Transactions on Electronic Computers, EC 15, No. 1, February 1966.

19. Seshu, S. and D. N. Freeman, "The Diagnosis of Asynchronous Sequential Switching Systems", IRE Transactions on Electronic Computers, August 1962.

20. Shen, V. Y., A. C. McKellar and P. Weiner, "A Fast Algorithm for Testing Switching Functions for Disjunctive Decompositions", Princeton Conference on Information Science.

21. Whitehouse, G. E. and A. A. B. Pritsker, "GERT: Part III - Further Statistical Results: Counters, Renewal Time, and Correlations", AIIE Transactions, Vol. 1, No. 1, March 1969.

22. Whitehouse, G. E. and L. Riccio, "Fault Detection in Logic Circuits Using GERT", Future Publication under NASA Grant -- Approximate Date, March 1970.

23. Whitehouse, G. E., "GERT, A Useful Technique for Analyzing Reliability Problem", to appear in February 1970 issue of Technometrics.

VITA

PERSONAL HISTORY

Name: Lucius Joseph Riccio

Date of Birth: August 26, 1947

Place of Birth: Bridgeport, Connecticut

Parents: Louis and Beatrice Riccio

EDUCATIONAL BACKGROUND

Trumbull High School
Trumbull, Connecticut — Graduated 1965

Lehigh University
Bachelor of Science Degree
Industrial Engineering — Graduated 1969

EXPERIENCE

Research Assistant
Lehigh University
Summer 1969

Industrial Engineer
Eastman Kodak Company
Rochester, New York
Summer 1968

Industrial Engineer Trainee
Carpenter Steel Company
Bridgeport, Connecticut
Summer 1967

MISCELLANEOUS

National Science Foundation Trainee for Graduate Study

Engineer-in-Trainee, State of Pennsylvania

Member, Alpha Pi Mu